MAXIMAL ARC-DISJOINT AND NODE-DISJOINT FLOW IN MULTICOMMODITY NETWORKS

A Thesis Submitted
In Partial Fulfilment of the Requirements
for the Degree of
MASTER OF TECHNOLOGY

By

R. K AHUJA

to the

INDUSTRIAL AND MANAGEMENT ENGINEERING PROGRAM

INDIAN INSTITUTE OF TECHNOLOGY, KANPUR

JULY, 1979

CERTIFICATE

This is to certify that the present work on 'MAXIMAL ARC-DISJOINT AND NODE-DISJOINT FLOW IN MULTICOMMODITY NETWORKS' by R. K. Ahuja has been carried out under my supervision and has not been submitted elsewhere for the award of a degree.

(A. K. Mittal)
Assistant Professor
Industrial and Management
Engineering Programme
Indian Institute of Technology
Kanpur 208016

July 1979.

IMEP-1979-M-AHU-MAX

ACKNOWLEDGEMENTS

I feel great pleasure in expressing my sense of gratitude to Dr. A. K. Mittal who suggested the problem and provided excellent guidance throughout the course of this work.

I am also greatly indebted to Dr. J. L. Batra for providing a helpful and sympathetic attitude which is essential for research work.

R. K. Ahuja

CONTENTS

SYNOPSIS

In this work the problem of determining the maximal arc-disjoint and node-disjoint flow in a multicommodity network is considered. This is the restriction of the well known multicommodity maximal flow problem with the added constraints that each arc allows flow of at most one commodity (arc-disjoint flow), or each node allows flow of at most one commodity (node-disjoint flow).

Both the problems for two-commodity directed networks are formulated as integer linear programming problems and methods are suggested to solve them. Branch and bound methods are developed to obtain optimal solutions, and heuristics to obtain approximate solutions. Computer programs are designed for each of the method suggested and tested over a large number of randomly generated networks. Computational performance of the programs on DEC-1090 time-sharing computer system is presented.

Arc-disjoint and node-disjoint flows in multicommodity networks are also considered. Two heuristics are developed for each of the problem. Computational performance of the heuristics over randomly generated problems is also presented.

CHAPTER I

INTRODUCTION

The aim of an operations researcher has always been to tackle the untackled problems, to solve the unsolved problems and to propose better solution procedures for the solved ones. This work is also consistent with this aim. I have tackled untackled problems, proposed methods to solve them and given hints for their improvements and extensions.

On account of its wide applicability, network flows are considered an important branch of study in the field of Operations Research. Problems in the study of airline systems, railroad systems, shipping systems and in general all transportation planning problems are network problems on appropriately defined networks. Problems in natural gas, crude oil or other fluid flows are problems in the appropriate pipeline networks. Economic models can also be treated as network models by representing factories, warehouses and markets as points and by treating highways, railroads and waterways, and other transportation channels as lines. Thus network algorithms find a vast number of applications in communication, transportation and distribution and in solving various practical problems that can be modelled as network flow problems.

If a network has many source nodes and many sink nodes and distinct commodities flowing between each source-sink pair, then it is a multicommodity network and flow problems over it are referred to as multicommodity network flow problems. Due to technological reasons if arcs allow only one commodity to flow over it, then any feasible flow will be arc-disjoint, i.e., commodities will flow over disjoint (mutually exclusive) sets of arcs. Similarly, node-disjoint flow can be defined as flow in which commodities flow over disjoint set of nodes.

The problem of determining maximal arc-disjoint and node-disjoint flow in a multicommodity network is considered in this work. The problem may be looked upon as to subdivide the set of arcs (or nodes) into as many subsets as the number of commodities. With each commodity flowing over its subset of arcs (or nodes), it is required to find that subdivision which ends up with maximum sum of flows. Evidently it is a combinatorial optimization problem with difficult solvable properties. One has to device special solution techniques to solve them.

Chapter II presents a state-of-art survey of results and algorithms for multicommodity flow problems. This survey is supplemented by an extensive list of references.

Chapter III considers edge-disjoint maximal flow problem in a two-commodity directed network. It describes

integer linear programming formulation of the problem. Details of a branch and bound algorithm developed to solve this problem optimally and of two heuristics developed to solve large sized problems are described. Computational performance of all the algorithms described in this chapter is also presented.

Chapter IV considers maximal node-disjoint flow in a two-commodity directed network. It describes the mathematical programming formulation of the problem, details of a branch and bound algorithm to solve the problem optimally and of two heuristics to solve large sized problems. In essence edge-disjoint flow problems and node-disjoint flow problems are similar problems, hence solution procedures for them are along similar lines.

Extensions of the heuristics developed for two-commodity networks to multicommodity networks is considered in Chapter V. Computer programs for them are also written and computational performances of the programs is discussed. Lastly, in Chapter VI avenues for further research are mentioned.

CHAPTER II

A REVIEW OF MULTICOMMODITY NETWORK FLOW PROBLEMS

This chapter aims at a comprehensive survey of the literature dealing with the multicommodity flow problems. The survey may logically be spread into three sections. Section 1 discusses the properties of multicommodity network flow problems. Section 2 presents a brief survey of special results. General multicommodity network flow problems and their solution procedures are discussed in Section 3.

2.1 GENERAL REMARKS

Multicommodity flow problems arise naturally in modelling wherever commodities, vehicles or messages are to be shipped or transmitted from certain nodes of an underlying network to some others. Considerations to simple examples make it clear that the multicommodity flow problems are considerably more complex than the single commodity one. The nice combinatorial features of the single commodity case are lost in its generalization. The constraint matrix is no longer unimodular hence optimal solutions are generally non-integer. The max-flow, min-cut theorem true for single

commodity networks is false for multicommodity networks and no simple minded modification of the labelling process seem to work. However, the constraint matrix has a special structure of being block angular which is exploited to produce efficient algorithms.

2.2 SPECIAL RESULTS

Most of the special results for multicommodity network flow problems are restricted to special networks or networks with at most two sources or two sinks or two or three commodities. These results provide little aid in attacking the general problems, however, their study is advantageous before proceeding to the general problems.

2.2.1 Two Commodity Flow Problems

Two-commodity flow in undirected networks enjoys many special properties. Max-flow, min-cut theorem is valid for it a hence optimal solutions are integer. Algorithms for constructing maximal two-commodity flow are efficient and may be found in [16,18,30].

Two-commodity flow in directed networks does not possess the properties of two-commodity flow in undirected networks. Optimal solutions are generally non-integer. It is harder to solve this class of problems, in fact, as difficult as linear programs.

2.2.2 Common Sink and Common Source Multicommodity Flow Problems

A multicommodity network, which has exactly one sink, i.e., all commodities terminate at one point, can be solved very efficiently due to its special structure. Max-flow, min-cut theorem is valid and hence optimal values are integer. Different commodities may be treated as a single commodity while solving the problem, and later flow values of each commodity may be identified by an arc-chain decomposition. Algorithm for finding maximal flow is presented in [29]. Minimum cost flow problems for these networks can also be solved efficiently employing the same ideas.

Common source multicommodity flow problems can be transformed to common sink multicommodity flow problems by a simple transformation and then all efficient algorithms available for common sink problems may be applied to it.

2.2.3 Miscellaneous Results

Rothfarb and Frish [28] present a max-flow, min-cut theorem and an algorithm for a special six-node, three commodity flow problem on a complete graph. Kleitman [25] presents an algorithm to obtain maximal flow when each node of the graph is a sink for all but one commodity. Recently Evans [7] has described sufficient condition by

which a certain class of multicommodity minimum cost network flow problem can be transformed and solved as an equivalent single commodity flow problem.

2.3 GENERAL MULTICOMMODITY FLOW PROBLEMS

This section discusses some important multicommodity flow problems and their solution procedures.

2.3.1 Multicommodity Network Realization Problem

The problems concerning that prescribed flow values of commodities can be realized in a given network or not come under this heading. These problems are encountered quite occasionally in practise and hence researchers have attempted to tackle it. One solution technique for this problem is presented in [16]. It is basically a revised simplex technique and it may solve moderately sized problems.

2.3.2 Network Synthesis Problem

In this problem we seek an optimal network configuration satisfying a given set of requirements. These problems may be viewed as specializations to network design problems. In general, we expect them to be more difficult since each tentative configuration may have to be analysed to find the optimal configuration. A survey of literature available for this class of problems is given in [36].

2.3.3 Multicommodity Maximal Flow problem

Determining maximum sum of flow of all commodities in a multicommodity network which satisfies the capacity constraints and node conservation constraints belongs to this class of problems. Ford and Fulkerson [9] were probably the first to formulate this problem as a linear program and propose solution techniques. They proposed the arc-chain formulation of the problem. The problem is solved by revised simplex method which requires knowledge of the basis only. The variable to enter the basis is determined by solving shortest path problem for each commodity. Unfortunately no computational results are available for this technique.

The same problem has been formulated as a node-arc formulation in [19] and solved by the method similar to that of arc-chain formulation. Grinold [14] has suggested a resource directive solution technique which is reported to have slow convergence properties.

2.3.4 Multicommodity Minimum Cost Flow Problem

The multicommodity minimal cost flow problem is to determine the minimum cost multicommodity flow through the network that meets the demand of each commodity, subject to (1) supply restrictions, (2) arc-capacity restrictions,

and (3) flow conservation at each transhipment node. This problem is a linear program and may be solved by the simplex method or one of its variant. However, real world problems are frequently of such size that direct application of simplex is prohibitive. There are three basic approaches that have been proposed for developing specialized techniques for these problems. They are price-directive decomposition, resource-directive decomposition and partitioning methods.

A price-directive decomposition procedure directs the coordination between a master program and each of several subprograms. The objective is to obtain a set of prices such that the combined solution for all subproblems yields an optimal for the original problem. The master program is solved by the revised simplex method with the subproblems used to test for optimality and select candidates for entering the basis of the master. The subproblems are single commodity minimum cost flow problems and can be solved efficiently. These procedures for the original problem and its generalizations may be found in [3,31,34,35].

A resource directive decomposition procedure, when applied to a multicommodity network flow problem having q commodities is to distribute the arc capacity among the individual commodities in such a way that solving q subproblems yields an optimal flow for the coupled problem.

At each iteration an allocation is made and q single commodity flow problems are solved. The sum of the capacities allocated to an arc over all commodities is equal to the arc capacity allocation is developed. Robacker [27] was the first one to propose these ideas, however, he did not indicate how this approach could be implemented. Geoffrion [10], Kennington and Shalaby [24] and Held, Wolf and Crowder [15] have presented techniques for implementing this approach.

Partitioning techniques are specialization of the simplex method where constraints are partitioned into two sets known as current constraints and secondary constraints. The secondary constraints are relaxed and objective function is minimized while only current constraints are active. If the optimal solution thus obtained satisfies the secondary constraints, it is an optimal one for the original problem. Otherwise those secondary constraints, which are not satisfied by the current optimal solution, are added to the set of current constraints. The current constraints which are not binding at the optimal solution are relaxed and steps are repeated.

The motivation to apply this technique is the fact that only a few capacity constraints are binding at optimality. The paper of Graves and Mcbride [12] deal with

primal technique while the work of Grigoriadis and White [13] is a dual technique.

It seems that field of arc-disjoint and node-disjoint flow problems remains untouched as none of the articles on this subject could be found in literature. This is the reason that literature survey on disjoint-flow problems was not possible.

CHAPTER III

MAXIMAL FLOW IN ARC-DISJOINT TWO-COMMODITY NETWORKS

In this chapter we shall discuss arc-disjoint two-commodity maximal flow problem in a directed network. The problem is a variation of the two-commodity maximal flow problem in the sense that it is desired that flow of both the commodities should be arc-disjoint. These type of problems are encountered when the same network (say a pipeline system) is used for simultaneous transportation of two commodities and the commodities are such that they can not flow on the same arc at same time.

The analysis of these problems is heavily dependent on following algorithms:

a) Single commodity maximal flow algorithm.

b) Maximal augmenting path algorithm

c) Branch and bound approach.

These algorithms are briefly described in Appendix A.

This chapter is divided into four sections. In section 1 we shall discuss the mathematical programming formulation of the problem as a network flow problem and as a 0-1 integer programming problem. A branch and bound

method suited for the problem is developed to get the optimal solution of the problem in section 2. As the size of the problem which can be solved by branch and bound method is limited, two heuristics are developed and described in section 3. Computational performances of the branch and bound method and of two heuristics for randomly generated networks is presented in section 4.

3.1 MATHEMATICAL PROGRAMMING FORMULATION

The network G(N,A) is a finite set of nodes N and a finite set of ordered pair of nodes, called arcs, A. The network is such that there is at most one directed arc between any pair of nodes. Each arc (i,j) has a number b_{ij} attached to it, called capacity, which denotes the maximum amount that can flow on the arc. Two distinct commodities flow in the network, with s^1, t^1 and s^2, t^2 being the source and sink node for commodity 1 and 2 respectively.

Let v^k denote the flow of commodity k between its source-sink pair and x_{ij}^k denote the flow value of commodity k on arc (i,j).

Objective function : The objective is to maximize the sum of flows of the two commodities and it may be expressed as:

$$\text{Max. } (v^1 + v^2) \qquad (3.1.1)$$

Constraints : There are three different sets of constraints for the problem.

Flow conservation constraints : These constraints essentially represent the fact that flow of each commodity is conserved at all nodes, except source and sink.

$$\sum_{j=1}^{n} x_{ij}^{k} - \sum_{j=1}^{n} x_{ji}^{k} = \begin{cases} v^k & \text{if } i = s^k \\ -v^k & \text{if } i = t^k, \; i = 1,2,\ldots,n \\ 0 & \text{otherwise, } k = 1,2 \end{cases} \qquad (3.1.2)$$

Capacity constraints : Sum of flows of both the commodities over an arc should be less than the capacity of that arc, which is expressed as:

$$\sum_{k=1}^{2} x_{ij}^{k} \leq b_{ij}, \; \forall (i,j) \in A \qquad (3.1.3)$$

Arc-disjoint constraints : Each arc allows flow of at most one commodity over it, which may be accounted as:

$$x_{ij}^{1} \cdot x_{ij}^{2} = 0, \quad \forall (i,j) \in A \qquad (3.1.4)$$

Non-negativity restrictions : Lastly there are restrictions that arc flow values of each commodity should be non-negative:

$$x_{ij}^{k} \geq 0, \quad \forall (i,j) \in A, \quad k = 1, 2 \qquad (3.1.5)$$

The arc-disjoint constraints in this formulation are non-linear constraints and usually are more difficult to

handle than the linear constraints. This formulation may be reduced to a linear integer programming formulation as follows:

Integer programming formulation: Let us introduce the 0-1 variables y_{ij}^k and a large positive number M. Then the arc-disjoint constraints may be written as:

$$x_{ij}^1 - M y_{ij}^1 \leq 0 , \quad \forall (i,j) \in A \qquad (3.1.6)$$

$$x_{ij}^2 - M y_{ij}^2 \leq 0 , \quad \forall (i,j) \in A \qquad (3.1.7)$$

$$y_{ij}^1 + y_{ij}^2 \leq 1 , \quad \forall (i,j) \in A \qquad (3.1.8)$$

$$y_{ij}^k = 0, 1 , \quad \forall (i,j) \in A, \; k = 1,2 \qquad (3.1.9)$$

On arc (i,j) flow of commodity 1 can be positive only if $y_{ij}^1 = 1$, which follows from (3.1.6). From (3.1.7) it follows that on the same arc flow of commodity 2 can be positive only if $y_{ij}^2 = 1$. From (3.1.8) it follows that only one commodity can flow arc (i,j).

For a network having n nodes and m arcs, this formulation will require 4m variables and 2n+4m constraints. It may be solved by any one of the integer programming algorithms, however solution of a moderately sized problem by such an approach may be prohibitive because of large computer memory requirement. Moreover, cost of obtaining

the optimal solution may be high because of slow convergence properties of these algorithms.

Hence there is a need for methods which overcome these limitations. The branch and bound algorithm, specially written for this class of problems, may be applied to it successfully. Details of the branch and bound algorithm are given below.

3.2 BRANCH AND BOUND ALGORITHM

The branch and bound algorithm seeks an optimal assignment of arcs to the two commodities. In this process of seeking it implicitly enumerates all possible assignments. It performs it systematically and aims to do it with least computations.

In the beginning all arcs allow both the commodities to flow over it simultaneously. An appropriate arc is chosen as the basis of branching and two subproblems are generated. One subproblem has the chosen arc assigned to commodity 1 and second to commodity 2. Feasibility of the subproblems is checked and subproblems found infeasible are added to the list of infeasible subproblems. A subproblem is taken out of this list and more subproblems are generated in similar fashion. Pruning and fathoming actively cut down the number of active subproblems. The

algorithm terminates when list of infeasible subproblems is empty. The strategies which characterize the branch and bound algorithm are described below:

Upper bounding strategy : Each subproblem has a unique subdivision of arcs into three following subsets:

S_1 : set of arcs allowing only commodity 1 to flow over it.

S_2 : set of arcs allowing only commodity 2 to flow over it.

S_0 : Set of arcs allowing both the commodities to flow over it simultaneously.

To obtain the upper bound for a subproblem solve single commodity maximal flow problem from s^1 to t^1 over the network which comprises of arcs $S_1 \cup S_0$. Repeat the same for second commodity over the network comprising of arcs $S_2 \cup S_0$. Let v^1 and v^2 denote the maximal flow values respectively. Then the value of the upper bound for the subproblem is $(v^1 + v^2)$.

This is a logical upper bound in the sense that any further assignment of unassigned arcs to the commodities shall reduce its value because maximal flow problems are solved over reduced networks.

Branching strategy : Branching is done at a subproblem. Using the arc flow values of the subproblem, which are the values obtained while determining the upper bound for the

subproblem, an arc is chosen which has simultaneous flow of both commodities and maximum sum of flows among such arcs. This arc serves as a basis of branching at the subproblem and two new subproblems are generated. One subproblem has the chosen arc assigned to commodity 1 and second subproblem has the chosen arc assigned to commodity 2.

The algorithm maintains a list of subproblems. The two new subproblems generated at this step are checked. If found infeasible, they are added to the end of the list in the order of increasing upper bounds. Feasible subproblems are used to improve the incumbent.

Searching strategy : This strategy finds out the subproblem at which branching is done. The last subproblem of the list of subproblems is selected to be branched at. This results in what commonly known as 'depth first search strategy'.

3.2.1 Stepwise description of the algorithm

Following is the stepwise description of the branch and bound algorithm for two-commodity edge-disjoint maximal flow problem:

Let

$L = \{s_1\}$ be the set of active infeasible subproblems

I denote the value of the incumbent

$U(s_1)$ denote the upper bound of the subproblem s_1.

Step 0 : Set I=0 and L to be empty.

Solve the single commodity maximal flow problem for commodity 1 over network G(N,A). Repeat the same for second commodity. Let v^1 and v^2 denote the maximum flow values obtained respectively.

If arc flow values are feasible i.e., no arc having two commodities flowing over it, stop, it is the optimal flow; otherwise let subproblem s_o represent these flow values with upper bound as $(v^1 + v^2)$.

Add s_o to list L. Go to step 1.

Step 1 : Check the list L. If it is empty go to step 6, otherwise take out the last subproblem of the list. Let it be s_m. Update the list L and go to step 2.

Step 2 : If upper bound of s_m is less than the incumbent then prune this subproblem and go to step 1, otherwise using the arc flow values of the subproblem, determine the arc which has flow of both commodities over it and maximum sum of the flows among such arcs. Go to step 3.

Step 3 : Generate two subproblems s_{m1} and s_{m2}. Subproblem s_{m1} has the arc selected in step 2 assigned to commodity 1 and subproblem s_{m2} has the selected

arc assigned to commodity 2. Determine the upper bounds for both subproblems and go to step 4.

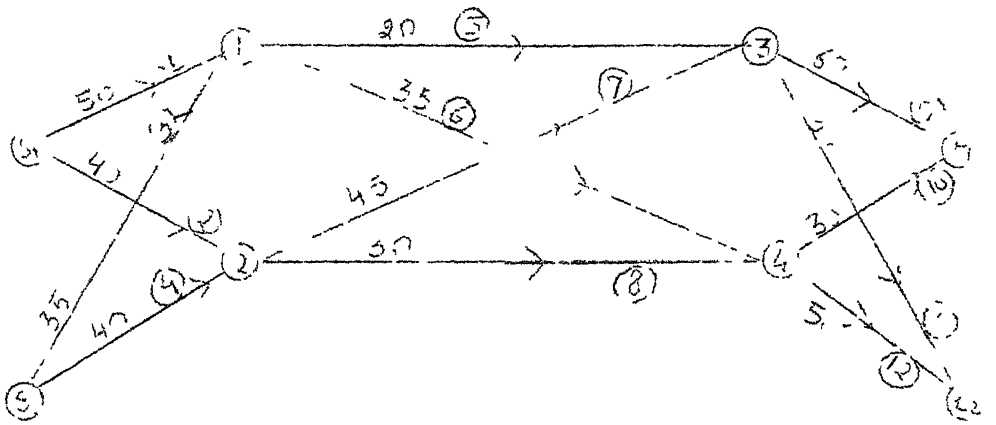
Step 4 : Check if subproblem s_{m1} has feasible flow values. If so go to step 5.
Similarly check for subproblem s_{m2}. Add the infeasible subproblems to the end of list L in increasing order of the upper bound.
Go to step 1.

Step 5 : If upper bound of the subproblem is greater than the value of the incumbent, then change the incumbent value, the new value equal to the upper bound of the subproblem.
Go to step 4.

Step 6 : Incumbent value is the maximal flow value. Stop.

3.2.2 Illustration

A small network flow problem is solved depicting steps of the branch and bound algorithm developed in this section.



NOTE: The arc number and arc capacity are mentioned over each arc as circled and uncircled numbers respectively.

ITERATION 1

Step 0 : Set I=0 and L to be empty. Solutions of the two s.c. maximal flow problems yields $v_1 = 90$ and $v_2 = 75$. Flow values are infeasible. Let these flow values correspond to subproblem (1) with upper bound as (90+75) = 165. This subproblem is added to list L.

Step 1 : Subproblem (1) is taken out of the list L. Go to step 2.

Step 2 : Upperbound of the subproblem is greater than incumbent. From the flow values at this subproblem the arc selected as a basis of branching is arc 7.

Step 3 : Two new subproblems labelled as (2) and (3) are generated. Subproblem (2) has arc 7 assigned to commodity 1 and subproblem (3) has arc 7 assigned to commodity 2. Upper bounds are found as U(2) = 155, U(3) = 125. Go to step 4.

Step 4 : Both the subproblems are found to be infeasible and are moved to list L in order of increasing upper bound. Go to step 1.

ITERATION 2

Step 1 : Subproblem with label as (2) is taken out of list L. Go to step 2.

Step 2 : Upper bound of this subproblem is greater than the incumbent. Arc 6 is selected as the basis of branching. Go to step 3.

Step 3 : Subproblem with label (4) and (5) are generated with upper bounds as U(4) = 140, U(5) = 125 respectively.

Step 4 : Both the subproblems are infeasible and hence are moved to list L in order of increasing upper bound. Go to step 1.

ITERATION 3

Step 1 : Subproblem (4) is taken out from list L. Go to step 2.

Step 2 : Upper bound of this subproblem is greater than incumbent. Arc 5 is selected as a basis of branching.

Step 3 : As the result of branching subproblem (6) and (7) are generated with upper bounds as U(6) = 120, U(7) = 120. Go to step 4.

Step 4 : Both the subproblems generated are feasible hence are not added to L. Go to step 5.

Step 5 : I = 120, Go to step 1.

ITERATION 4

Step 1 : Subproblem (5) is taken out of list L. Go to step 2.

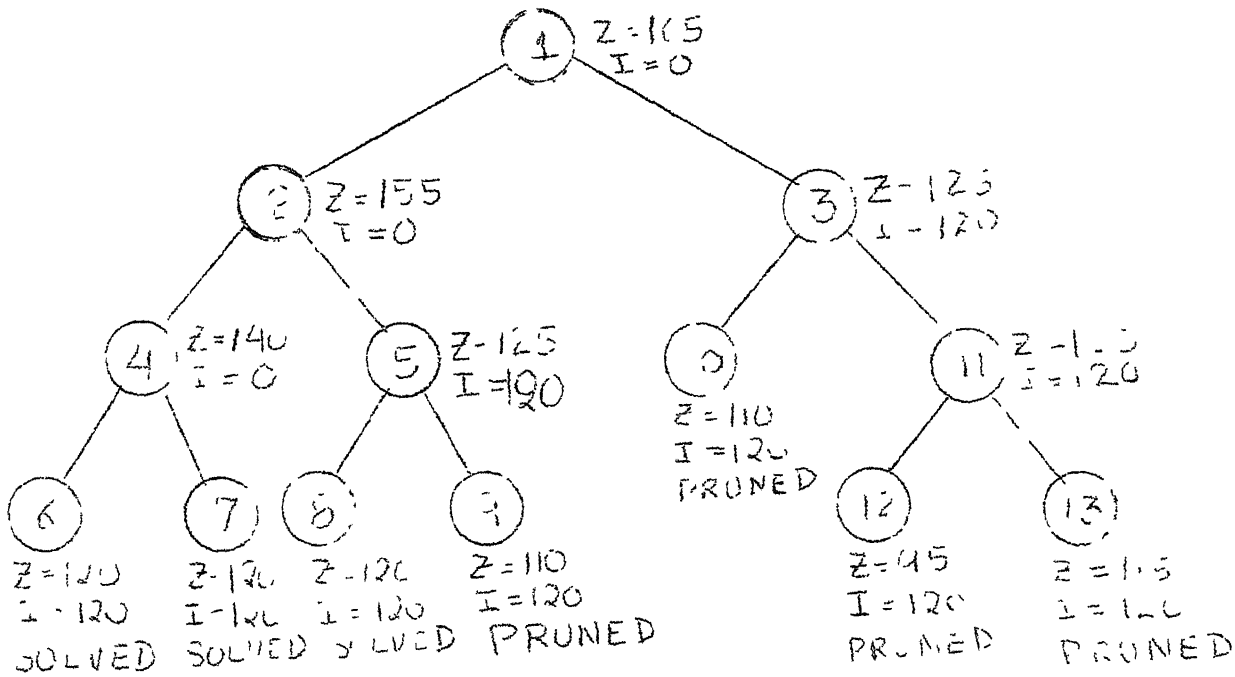
Step 2 : Upper bound of the subproblem is greater than incumbent. Arc 5 is selected as a basis of branching. Go to step 3.

Step 3 : Subproblem (8) and (9) are generated as a result of branching with upper bounds as U(8) = 120, U(9) = 110.

Step 4 : Both the subproblems are feasible. Go to step 5.

Step 5 : I = 120, Go to step 1.

This way steps are to be followed. The branch and bound tree developed by the algorithm for this problem is as follows:



3.3 HEURISTIC METHODS

The branch and bound method developed for the two-commodity arc-disjoint maximal flow problem can solve a moderately sized problem in a reasonable amount of time. Solutions of a large network flow problem by this approach may be prohibitive due to large computer memory requirement and thus a need arises for the methods to solve large problems. Also there may be situations where only a good solution is required and search for the optimal solution may not be necessary. Moreover, a tradeoff between computer resources, i.e., time and memory, and the optimality of the solution may suggest that recourse may be taken to the heuristic methods.

In this section we shall develop two heuristic methods which are intuitively appealing and are easy to implement.

3.3.1 Heuristic 1

The idea behind this heuristic is very simple and yet very effective. Two single commodity maximal flow problems are solved between s^1 to t^1 and s^2 to t^2 respectively. The arc which has flow of both the commodities and has maximum sum of flows among such arcs is selected. Decrease in the value of the maximal flow for each commodity is calculated if the selected arc is removed from the network.

Finally, the selected arc is assigned to the commodity for which decrease in the maximal flow is larger. Flow values are changed accordingly and the steps are repeated. When no arc is left which has flow of both commodities over it, the method terminates.

The detailed step by step procedure is given below:

NOTATIONS

S_1 : Set of arcs assigned to commodity 1.

S_2 : Set of arcs assigned to commodity 2.

S_0 : Set of unassigned arcs.

$\emptyset$: Empty set

Step 0 : Set $S_1 = S_2 = \emptyset$

and $S_0 = A$.

Go to step 1.

Step 1 : Solve s.c. max. flow problem from s^1 to t^1 over $G(N,A)$. Let $X^1 = \{x^1_{ij}\}$ denote the arc flow vector and v^1 the maximal flow value.

Repeat the same for commodity 2.

Let $X^2 = \{x^2_{ij}\}$ and v^2 denote the respective flow values. Go to step 2.

Step 2 : Using the arc flow values X^1 and X^2, select the arc (k,l) such that $x^1_{kl} > 0$, $x^2_{kl} > 0$, and $(x^1_{kl}+x^2_{kl})$ is maximum among all such arcs. If no such arc is found go to step 7; otherwise go to step 3.

Step 3 : Solve the s.c. max. flow problem from s^1 to t^1 over $G(N,A')$, where $A' = S_0 \cup S_1 - (k,1)$. Let X^1_* and v^1_* denote the arc flow vector and maximal flow values respectively. Similarly do it for second commodity from s^2 to t^2 over $G(N,A'')$ where $A'' = S_0 \cup S_2 - (k,1)$. Let X^2_* and v^2_* be the respective flow values. Go to step 4.

Step 4 : If $(v^1 - v^1_*) \geq (v^2 - v^2_*)$, go to step 5; otherwise go to step 6.

Step 5 : Assign the arc $(k,1)$ to commodity 1.

$S_1 = S_1 + (k,1)$

$S_0 = S_0 - (k,1)$

$X^2 = X^2_*$

$v^2 = v^2_*$

Go to step 2.

Step 6 : Assign the arc $(k,1)$ to commodity 2.

$S_2 = S_2 + (k,1)$

$S_0 = S_0 - (k,1)$

$X^1 = X^1_*$

$v^1 = v^1_*$

Go to step 2.

Step 7 : The flow values X^1, X^2, v^1 and v^2 are the optimal flow values. Stop.

3.3.2 Heuristic-2

The underlying thumb rule in this heuristic is that augmenting the flow along maximal augmenting path yields more flow. Definition of the maximal augmenting path and an algorithm to find such a path is given in Appendix A. Procedure starts by determining maximal augmenting path from s^1 to t^1 and similarly from s^2 to t^2. Flow is augmented along the path which has more augmenting capacity. All the arcs that lie on the path selected for augmentation are assigned to the commodity which flows over it. Contrary to heuristic-1 more than one arcs are assigned to a commodity in one iteration. With the changed flow values, maximal augmenting paths are determined for each commodity and path with more augmenting capacity is augmented by the respective commodity. This step is repeated until no augmenting path exists from s^1 to t^1 and from s^2 to t^2.

A detailed step by step procedure is given below:

The sets S_1, S_2 and S_0 have the same meaning as in the description of Heuristic 1.

Step 0 : Set $S_1 = S_2 = \emptyset$, $S_0 = A$.
Go to step 1.

Step 1 : Determine maximal augmenting path from s^1 to t^1 over the network which comprises of arcs $S_1 \cup S_0$.
Let a_1 denote the augmenting capacity of this path.
Go to step 2.

Step 2 : Determine maximal augmenting path from s^2 to t^2 over the network which comprises of arcs $S_2 \cup S_0$. Let a_2 denote the augmenting capacity of this path. Go to step 3.

Step 3 : If $a_1 = 0$ and $a_2 = 0$, go to step 6; otherwise determine the larger of the two numbers.
If $a_1 \geq a_2$ go to step 4, otherwise go to step 5.

Step 4 : Augment the flow on the maximal augmenting path of commodity 1. Let S_3 denote the set of arcs lying on this path. Then,

$$S_1 = S_1 \cup S_3$$
$$S_0 = S_0 - S_3$$

Go to step 1.

Step 5 : Augment the flow on the maximal augmenting path of commodity 2. Let S_3 denote the set of arcs lying on this path. Then

$$S_2 = S_2 \cup S_3$$
$$S_0 = S_0 - S_3$$

Go to step 1.

Step 6 : The flow values are optimal flow values.
Stop.

3.4 COMPUTATIONAL PERFORMANCE

The branch and bound algorithm and both the heuristics, described in previous sections of this chapter, were coded in FORTRAN-IV for DEC-1090 time-sharing computer system. The computer programs were tested over a large number of random networks generated as follows:

The node-node adjacent matrix, which specifies a network, was generated randomly. For each position of the adjacent matrix a random number, uniformly distributed over interval 0 to 1, was generated. If the number was found to be less than a prespecified number, then 1 was placed in that position, otherwise 0 was placed there. Some other considerations were taken into account which insured that the network would be connected graph and there would be at most one directed arc between any pair of nodes. Positions for sources and sinks were generated in a similar manner. Parameters of a randomly generated network are number of nodes in the network and its density. With number of nodes fixed, higher density means more arcs. The relationship between number of nodes (n), number of arcs (m) and density (d) is

$$d = \frac{2m}{n^2}$$

At the end of each computational run two time values are available -- CPU time and elapsed time. CPU (Central Processing Unit) time provides the time that CPU spent to complete the computations, whereas elapsed time provides the time during which the programme was in the state of execution. DEC-1090 system is a multiprogramming system and a programme remains in the state of execution for a relatively long time while CPU processes it only intermittantly. Therefore elpased time is generally more than the CPU time depending upon the load of the system. This is the reason that CPU time is considered the right estimate of computational time.

Two types of experiments were conducted. First type to measure the computational time of the computer programs of all the algorithms and second type to measure the deviations of the heuristics from the optimal solution. These were conducted for several combinations of the parameters, with number of nodes varying from 10 to 100 and density varying from 0.10 to 0.80. The branch and bound algorithm could solve problems having 50 nodes and 200 arcs, whereas heuristics could solve problems having 100 nodes and 2000 arcs.

To conduct first type of experiment 10 randomly generated problems were solved for each combination of the

parameters by all algorithms and CPU time was observed. The results of the experiment are tabulated in Table 3.4.1. Three figures in each square of the table represent the average CPU time in seconds to solve one problem by branch and bound algorithm, heuristic-1 and heuristic-2 respectively. To conduct second type of experiment 10 problems were solved for each combination of the parameters. Results of this experiment are tabulated in Table 3.4.2. This table provides percent problems whose maximal flow value deviated from optimal flow value by 0%, 0-5%, 5-10% ..., 20-25% or more, when solved by heuristic-1 and heuristic-2.

To derive more meaningful results from the tables and to get the feel of changes in the computational times, several graphs were plotted. In Fig. 3.4.1 the computational time is plotted vs. density for all the edge-disjoint algorithms. This graph shows the variation in times as the network becomes more dense. Figs. 3.4.2, 3.4.3 and 3.4.4 demonstrate the relationship between computational time, density and nodes for each edge-disjoint algorithm.

3.4.1 Performance of the branch and Bound Algorithm

The number of feasible solutions for an arc-disjoint flow problem are proportional to a^m. Thus as the problem size increases, the number of feasible solutions increase

enormously. Even though the percentage of feasible solutions enumerated by the branch and bound algorithm decreases with increase in problem size, the total enumerated solutions still increase rapidly. This drawback makes this uneconomical for large problems. However small or moderate sized problems are solved very efficiently. If the number of arcs in a network are less than 100 then it is generally solved in less than 5 seconds. So we recommend the use of the branch and bound algorithm for small and moderate sized problems.

3.4.2 Performances of the Heuristics

Normally computational time and accuracy of the solution are the criteria which measure the effectiveness of the heuristic programmes. Both the heuristics in this work measure well on these scales. Their efficiency in solving large problems within a reasonable time recommend their application. Problems having 1000 arcs are generally solved in less than 5 seconds.

Both the heuristics provide quite accurate solutions. At least 60 percent of the solutions provided by heuristic-1 were the optimal ones and in 90 percent of the cases deviations from optimal solutions were within 5 percent. None of the problems deviated by more than 20 . Heuristic-2

also provides quite good solutions, but not as good as heuristic-1.

Generally heuristic-2 solves a problem in less time than heuristic-1. However the difference in the time is not great. To summarize, we recommend the use of branch and bound algorithm for small and moderate sized problems, and application of heuristic-1 for large sized problems.

Table 3.4.1: Computational times for edge-disjoint algorithms (Times are in seconds)

No. of Nodes \ Density	0.10	0.15	0.20	0.30	0.50	0.80
10	-	-	0.01	0.02	0.05	0.25
	-	-	0.01	0.02	0.02	0.07
	-	-	0.01	0.02	0.02	0.04
15	-	0.02	0.02	0.05	0.45	0.76
	-	0.02	0.02	0.02	0.14	0.40
	-	0.02	0.02	0.03	0.07	0.13
20	0.03	0.03	0.05	0.12	0.53	2.30
	0.02	0.03	0.04	0.09	0.45	0.79
	0.03	0.04	0.04	0.08	0.15	0.22
25	0.04	0.05	0.18	3.03	6.58	-
	0.04	0.04	0.09	0.18	0.95	2.53
	0.04	0.06	0.09	0.12	0.24	0.58
30	-	-	-	-	-	-
	0.06	0.09	0.13	0.57	1.02	3.05
	0.07	0.09	0.14	0.24	0.47	0.89
50	-	-	-	-	-	-
	0.25	0.66	0.86	1.24	2.35	-
	0.35	0.46	0.64	1.07	2.31	-
80	-	-	-	-	-	-
	1.37	1.59	2.10	3.88	-	-
	1.03	1.63	2.53	4.18	-	-
100	-	-	-	-	-	-
	1.54	2.55	3.45	-	-	-
	2.35	3.89	4.55	-	-	-

Table 3.4.2: Performance of edge-disjoint heuristics

No. of nodes	Density	Heuristic-1: Percent problems with deviation from optimal as: 0	5	10	15	20	25	More	Heuristic-2: Percent of problems with deviation from optimal as: 0	5	10	15	20	25	More
10	0.2	100							100						
10	0.3	100							100						
10	0.4	100							100						
10	0.5	100							100						
10	0.8	60	30						90		10				
15	0.2	100							100						
15	0.3	100							90				10		
15	0.4	80		10	10				80	10	10				
15	0.5	90	10						100						
15	0.8	60	40						50	50	10	10			
20	0.1	100							100						
20	0.2	90		10					90			10			
20	0.3	90	10						100						
20	0.4	100							70		10	10			10
20	0.5	80	20						80	10		10			
25	0.1	100							100						
25	0.2	90	10						80	10	10				
25	0.3	70	20			10			40	50	10			10	10
25	0.4	80	20						80	10			10		
30	0.1	100							100						
30	0.2	90	10						90	10					

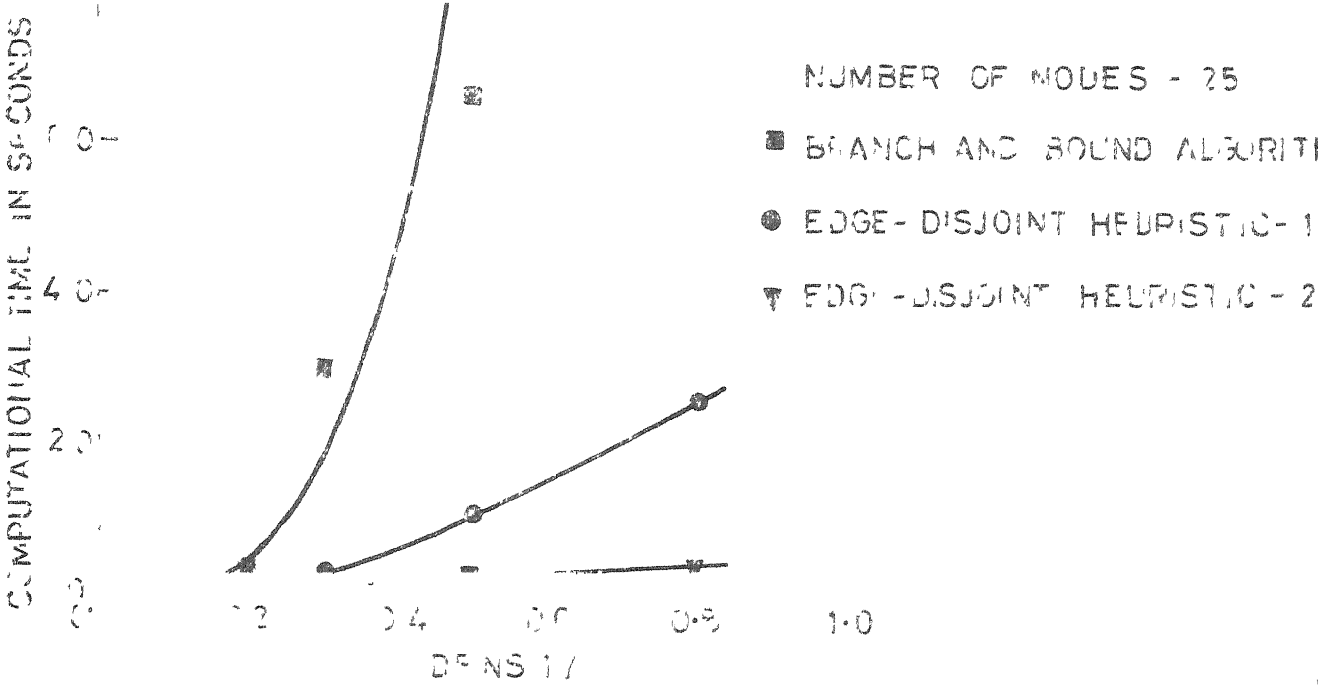
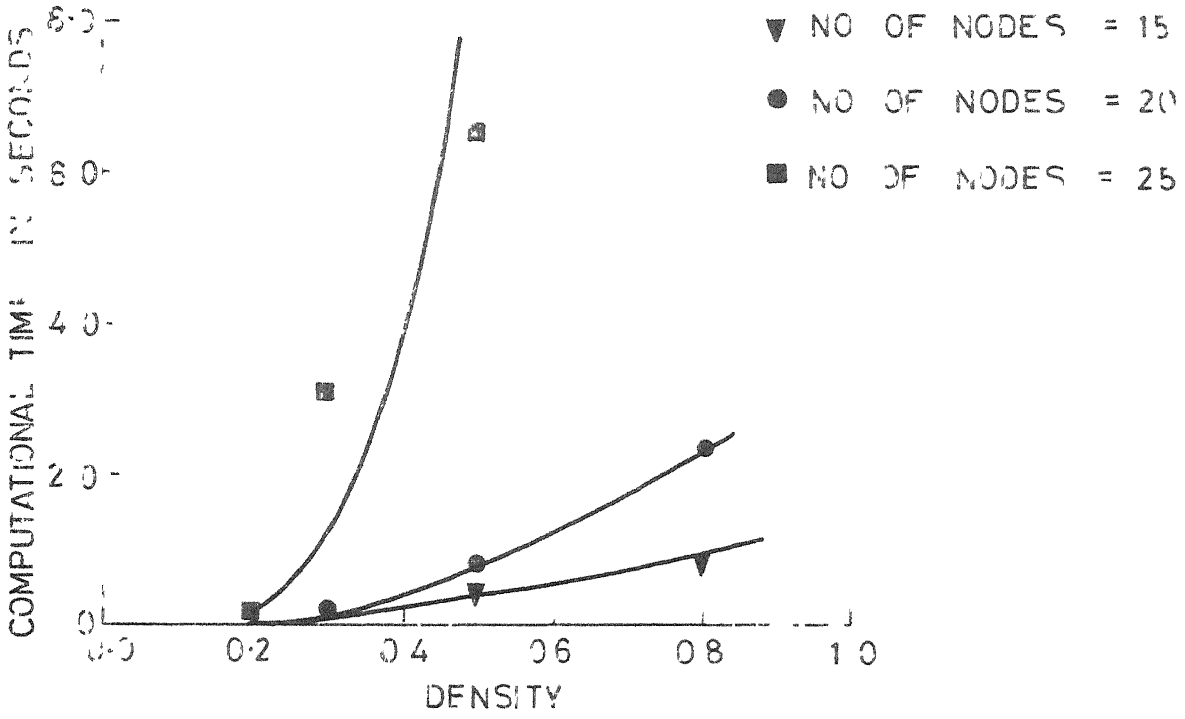


FIG. 3·4·1 RELATIVE PERFORMANCE OF ALGORITHMS



FIG·3·4·2 PERFORMANCE OF BRANCH AND BOUND ALGORITHM

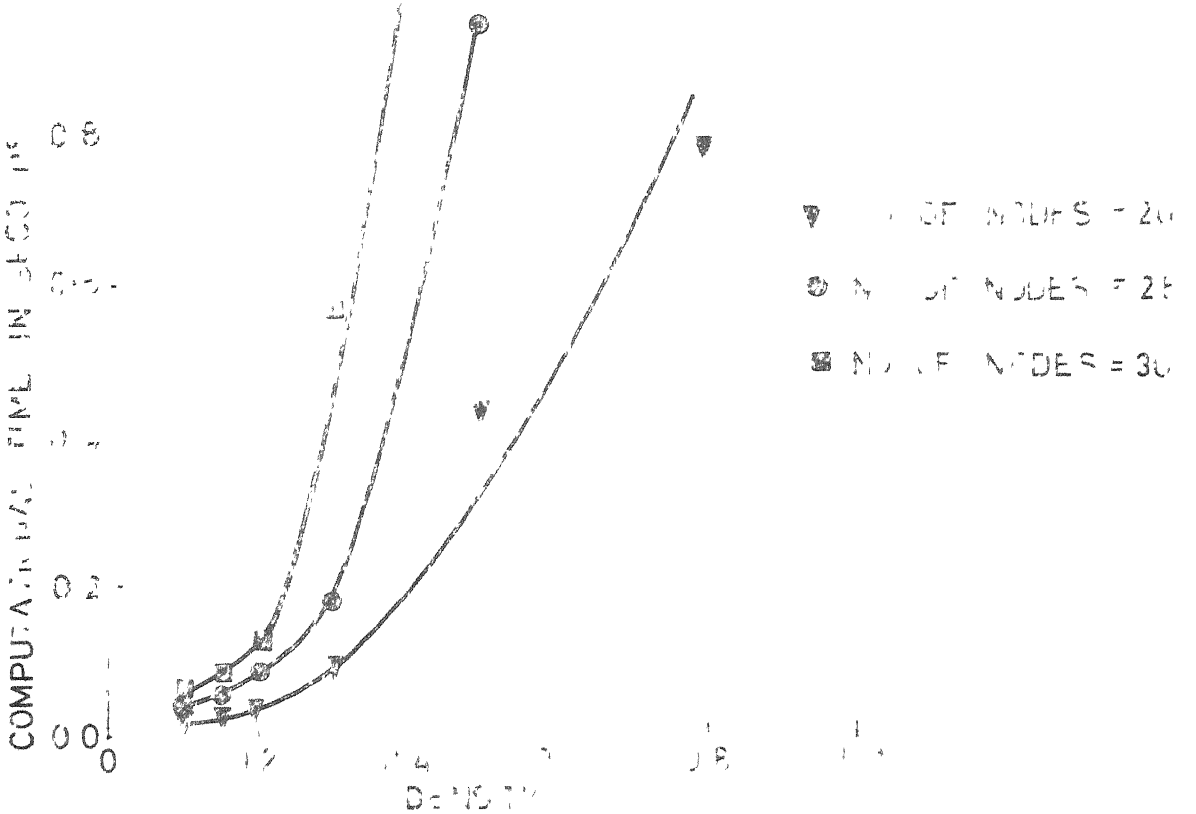


FIG. 3·4·3 PERFORMANCE OF EDGE-DISJOINT HEURISTIC-1

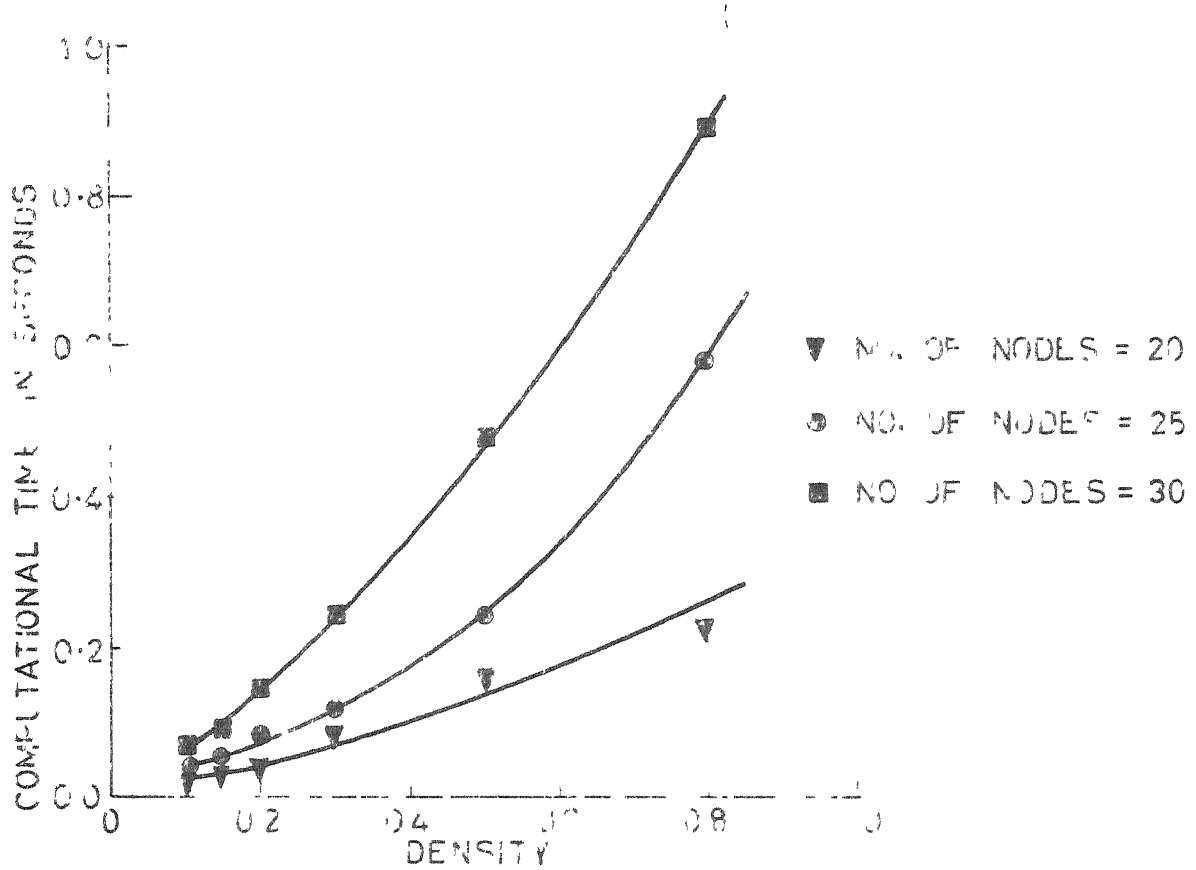


Fig.3·4·4 PERFORMANCE OF EDGE DISJOINT HEURISTIC-2

CHAPTER IV

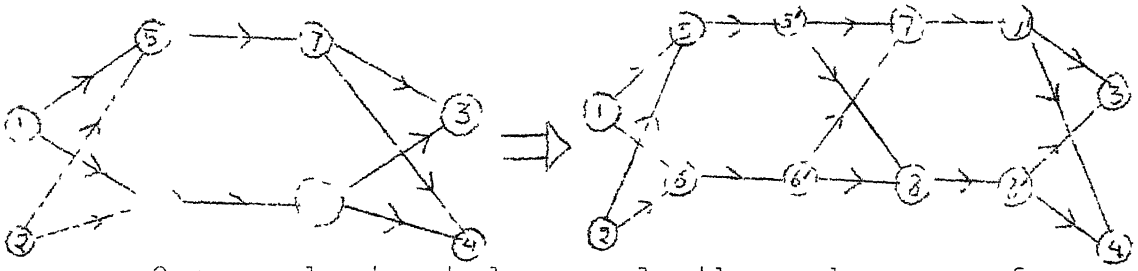
MAXIMAL FLOW IN NODE-DISJOINT TWO-COMMODITY NETWORKS

This chapter considers another variation of the two-commodity maximal flow problem. Additional constraints are added to the general problem that no node allows simultaneous flow of two commodities. The resulting flow pattern is node-disjoint in the sense that both commodities flow over disjoint set of nodes.

A result which is immediately available from the study of node-disjoint flow is that it is arc-disjoint also, which means that node-disjoint flow is a further restriction of arc-disjoint flow. Moreover all node-disjoint flow problems may be converted to arc-disjoint flow problems by a simple transformation described below.

For each node which has indegree and outdegree strictly greater than one, add one new node, which is connected with the former by a directed arc of large capacity such that all incoming nodes are incident upon the former and all outgoing on the latter. Other nodes need not be disturbed. Thus the transformed network, in general, will have more nodes and more arcs than the original network.

This transformation is best understood by a simple illustration given below:



One may be tempted to apply the techniques of arc-disjoint flow problem to the transformed network. However, a little reflection will reveal that it may not be desirable to do so, because techniques specially developed for node-disjoint flow will solve a problem in much less time than the application of arc-disjoint flow techniques to the transformed network. Reason is simple - number of nodes are generally less than the number of arcs and an algorithm whose basis is assignment of nodes will yield the solution in much less time than the algorithm whose basis is assignment of arcs.

This chapter follows the same pattern as that of Chapter 3. Section 1 discusses the mathematical programming formulation of the two-commodity node-disjoint maximal flow problem and an integer programming formulation for the same. A branch and bound algorithm developed for this problem is discussed in section 2. Section 3 describes the two heuristic approaches for the same problem and section 4 presents the computational performances of the algorithms developed in previous sections.

4.1 MATHEMATICAL PROGRAMMING FORMULATION

Objective Function : The objective is to maximize the sum of flows of the two commodities which may be expressed as:

$$\text{Max. } (v^1 + v^2) \qquad (4.1.1)$$

Constraints : There are three different sets of constraints for this problem.

Flow conservation constraints: These constraints represent the fact that flow of each commodity is conserved at all nodes except source and sink. This may be written as

$$\sum_{j=1}^{n} x_{ij}^k - \sum_{j=1}^{n} x_{ji}^k = \begin{cases} v^k & \text{if } i = s^k \\ -v^k & \text{if } i = t^k, \\ 0 & \text{otherwise} \end{cases} \quad i=1,2,\ldots,n, \; k = 1,2 \qquad (4.1.2)$$

Capacity constraints : Sum of flows of both the commodities over an arc should be less than the capacity of that arc:

$$\sum_{k=1}^{2} x_{ij}^k \leqslant b_{ij}, \quad \forall (i,j) \in A \qquad (4.1.3)$$

Node-disjoint constraints: Each node allows at most one commodity to flow through it. It may be accounted as:

$$\sum_{j=1}^{n} x_{ij}^1 \cdot \sum_{j=1}^{n} x_{ij}^2 = 0 , \quad i = 1, 2, \ldots, n \qquad (4.1.4)$$

Non-negativity restrictions : Lastly there are restrictions that arc flows of each commodity should be non-negative:

$$x_{ij}^k \geqslant 0, \quad \forall (i,j) \in A, \; k = 1,2 \qquad (4.1.5)$$

The node-disjoint constraints of this formulation are non-linear in nature and difficult to handle. They may be transformed to linear constraints as follows:

Integer programming formulation: Let the 0-1 variables y_i^k and a large positive number M be introduced. Then the node-disjoint constraints may be written as:

$$\sum_{j=1}^{n} x_{ij}^1 - M y_i^1 \leq 0, \quad i = 1,2,\ldots,n \qquad (4.1.6)$$

$$\sum_{j=1}^{n} x_{ij}^2 - M y_i^2 \leq 0, \quad i = 1,2,\ldots,n \qquad (4.1.7)$$

$$y_i^1 + y_i^2 \leq 1, \quad i = 1,2,\ldots,n \qquad (4.1.8)$$

$$y_i^k = 0, 1, \quad i = 1,\ldots,n, \; k = 1,2 \qquad (4.1.9)$$

From constraints in (4.1.6) it follows that flow of commodity 1 through any node N_i can be positive only if $y_i^1 = 1$. Constraints in (4.1.7) impose the same restriction for commodity 2. From constraints in (4.1.8) it follows that at most one commodity can flow through node N_i.

If a network which has n nodes and m arcs and is formulated in the way described above, then it will require (2n+2m) variables and (5n+m) constraints. This formulation can be solved by any one of the integer programming algorithms, however solution of a large problem by this approach may be prohibitive due to large number of variables and constraints.

Moreover cost of obtaining the optimal solution may be high because of slow convergence properties of these algorithms.

The branch and bound method, specially written for this class of problems overcomes these limitations and solves a moderately large problem in a reasonable amount of time.

4.2 BRANCH AND BOUND ALGORITHM

The branch and bound algorithm seeks an optimal subdivision of nodes to the two commodities. Initially all nodes allow simultaneous flow of both the commodities through it. An appropriate node is selected and two subproblems are generated. One subproblem has the selected node assigned to commodity 1 and the second subproblem has the selected node assigned to commodity 2. Feasibility of the subproblems is checked and infeasible ones are added to a list of subproblems. An appropriate subproblem is taken out of this list and more subproblems are generated. Pruning and fathoming actively cut down the number of subproblems. The algorithm terminates when list of infeasible subproblems is empty. Following are the strategies which characterize the brand and bound algorithm.

Upper bounding strategy : Each subproblem has a unique subdivision of nodes into three following subsets:

S_1 : Set of nodes allowing only commodity 1 to flow through it.

S_2 : Set of nodes allowing only commodity 2 to flow through it.

S_0 : Set of nodes allowing both the commodities to flow through it.

To obtain the upper bound at a subproblem solve single commodity maximum flow problem for commodity 1 over the network comprising of the nodes $S_1 U S_0$ and arcs that connect these nodes. Let v^1 represent the maximum flow value. Repeat the same for commodity 2 over the network comprising of nodes $S_2 U S_0$ and arcs connecting these nodes. Let v^2 represent the maximum flow value. Then the value of the upper bound for the subproblem is $(v^1 + v^2)$.

Branching strategy : Branching is done at a subproblem. Using the arc flow values of the subproblems, which are the values obtained while determining the upper bound for it, a node is selected which allows simultaneous flow of both the commodities and maximum sum of flows among such nodes. This node serves as a basis of branching and at a result two subproblems are generated. One subproblem has the selected node assigned to commodity 1 and the second subproblem has the selected node assigned to commodity 2.

The algorithm maintains a list of subproblems. The two new subproblems generated at this step are checked. If found infeasible, they are added to the list in the order of increasing upper bound. Feasible subproblems are used to improve the incumbent.

Searching strategy: This strategy finds out the subproblem at which branching is done. The last subproblem of the list of problems is selected for further branching.

4.2.1 Stepwise description of the algorithm

Following is the stepwise description of the branch and bound algorithm for two-commodity node-disjoint maximal flow problem.

Notations are same as described in section 3.2.1.

Step 0 : Set I = 0 and L to be empty.

Solve the single commodity maximal flow problem for commodity 1 over the network G(N,A). Repeat the same for commodity 2. Let v^1 and v^2 respectively denote the maximum flow values obtained. If arc flow values are feasible, i.e., no node is having two commodities flowing through it simultaneously, then stop as it is the optimal flow; otherwise let subproblem s_0 represent these flow values with upper bound as $(v^1 + v^2)$.

Add s_0 to L and go to step 1.

Step 1 : Check the list L. If it is empty go to step 6; otherwise take out the last subproblem of the list. Let it be s_m. Update the list L and go to step 2.

Step 2 : If upper bound of s_m is less than the incumbent then prune this subproblem and go to step 1; otherwise using the flow values of the subproblem determine the node which has flow of both the commodities over it and maximum sum of flows among such nodes. Go to step 3.

Step 3 : Generate two subproblems s_{m1} and s_{m2}. Subproblem s_{m1} has the node selected in step 2 assigned to commodity 1 and subproblem s_{m2} has the selected node assigned to commodity 2. Determine the upper bounds for both subproblems and go to step 4.

Step 4 : Check if subproblem s_{m1} has feasible flow values. If so, go to step 5.
Similarly check for subproblem s_{m2}. Add the infeasible subproblems to the end of list L in increasing order of the upper bounds. Go to step 1.

Step 5 : If upper bound of the subproblem is greater than the value of the incumbent, then change the incumbent value, the new value equal to the upper bound of the subproblem. Go to step 4.

Step 6 : Incumbent value is the maximal flow value.
Stop.

4.3 HEURISTIC METHODS

The branch and bound method developed for the two commodity node-disjoint maximal flow problem can solve problems having 50 nodes and 300 arcs. However, for larger problem size exponentially increasing computational time and large memory requirement prohibit the application of this approach. A recourse is sought through heuristics to solve such problems. Two heuristics which are along the same lines as their edge-disjoint counterparts are proposed. Their detailed discussion follows now.

4.3.1 Heuristic-1

Two single commodity maximal flow problems are solved from s^1 to t^1 and s^2 to t^2 respectively. The node which has flow of both the commodities through it and has maximum total flow among such nodes is chosen. Decrease in the value of the maximal flow for each commodity is calculated if the chosen node and arcs incident upon it are removed from the network. The chosen node is finally assigned to the commodity for which decrease in the maximal flow is larger With the changed flow values, which occur as a result of the assignment, another node is selected employing the same criteria. The method terminates when no node is left which has flow of both commodities through it.

Following step by step procedure may be suggested for this methodology.

NOTATIONS

S_1 : Set of nodes assigned to commodity 1.

S_2 : Set of nodes assigned to commodity 2.

S_0 : Set of unassigned nodes.

Step 0 : Set $S_1 = S_2 = \emptyset$, and $S_0 = A$.

Go to step 1.

Step 1 : Solve s.c. max. flow problem from s^1 to t^1 over $G(N, A)$. Let $X^1 = \{x^1_{1J}\}$ denote the arc flow vector and v^1 the maximal flow value.

Repeat the same for second commodity. Let $X^2 = \{x^2_{1J}\}$ denote the arc flow vector and v^2 the maximal flow value.

Go to step 2.

Step 2 : Using the arc flow vector X^1 and X^2 select the node n_o such that $\sum_{J=1}^{n} x^1_{n_o J} > 0$, $\sum x^2_{n_o J} > 0$ and $(\sum_{J=1}^{n} x^1_{n_o J} + \sum_{J=1}^{n} x_{n_o J})$ is maximum among such nodes.

If no such node is found go to step 7; otherwise go to step 3.

Step 3 : Solve the s.c. max. flow problem from s^1 to t^1 over the network which comprises of nodes $S_1 \cup S_0 - \{n_o\}$ and arcs connecting these nodes. Let X^1_* and v^1_* denote the arc flow vector and maximal flow value respectively. Similarly do it for second commodity from s^2 to t^2 over the network which comprises of

nodes $S_2 \cup S_0 - n_o$ and arcs connecting these nodes. Let X^2_* and v^2_* denote the respective flow values. Go to step 4.

Step 4 : If $(v^1 - v^1_*) \geq (v^2 - v^2_*)$ go to step 5, otherwise go to step 6.

Step 5 : Assign the node n_o to commodity 1.

$S_1 = S_1 + \{n_o\}$

$S_0 = S_0 - \{n_o\}$

$X^2 = X^2_*$

$v^2 = v^2_*$

Go to step 2.

Step 6 : Assign the node n_o to commodity 2.

$S_2 = S_2 + \{n_o\}$

$S_0 = S_0 - \{n_o\}$

$X^1 = X^1_*$

$v^1 = v^1_*$

Go to step 2.

Step 7 : The flow values X^1, X^2, v^1 and v^2 are feasible, hence stop.

4.3.2 Heuristic-2

Procedure starts by determining maximal augmenting path from s^1 to t^1 and similarly from s^2 to t^2. Flow is augmented along the path which has more augmenting capacity. All the nodes that lie on the path selected for augmentation

are assigned to the commodity which flows over it. With the changed flow values this basic step is repeated again. When no augmenting path with positive augmenting capacity is left from s^1 to t^1 and from s^2 to t^2, the procedure terminates.

The sets S_1, S_2 and S_0 have the same meaning as in the description of heuristic-1 in section 4.2.1.

Step 0 : Set $S_1 = S_2 = \emptyset$, and $S_0 = A$
Go to step 1.

Step 1 . Determine maximal augmenting path from s^1 to t^1 over the network which comprises of nodes $S_1 \cup S_0$ and arcs which connect these nodes. Let a_1 denote the augmenting capacity of this path. Go to step 2.

Step 2 : Determine maximal augmenting path from s^2 to t^2 over the network which comprises of nodes $S_2 \cup S_0$ and arcs connecting these nodes. Let a_2 denote the augmenting capacity of this path. Go to step 3.

Step 3 : If $a_1 = 0$ and $a_2 = 0$ go to step 6. Otherwise determine the larger of the two numbers a_1 and a_2. If $a_1 \geq a_2$, go to step 4; otherwise go to step 5.

Step 4 : Augment the flow on the maximal augmenting path of commodity 1. Let S_3 denote the set of nodes lying on the path. Then

$S_1 = S_1 \cup S_3$

$S_0 = S_0 - S_3$

Go to step 1.

Step 5 : Augment the flow on the maximal augmenting path of commodity 2. Let S_3 denote the set of nodes lying of the path. Then

$S_2 = S_2 \cup S_3$

$S_0 = S_0 - S_3$

Go to step 1.

Step 6 : Print the flow values and stop.

4.4 COMPUTATIONAL PERFORMANCE

The branch and bound algorithm and both the heuristics described in previous sections of this chapter were coded in FORTRAN-IV for DEC-1090 time-sharing computer system. These programs were tested over a large number of randomly generated networks. Computational times were measured in terms of CPU times. The method to generate random networks and justification of CPU time as a measure of time performance of algorithms is described in section 3.4.

Two types of experiments were conducted. Of first type to measure the computational time of the algorithms for several combinations of the parameters and second to measure the deviations of the heuristics from the optimal solution. The number of nodes varied from 10 to 100 and

density varied from 0.10 to 0.80. The branch and bound algorithm can solve problems having 50 nodes and 500 arcs and heuristics can solve problems having 100 nodes and 2000 arcs.

To conduct first type of experiment 10 problems were solved for each combination of the parameters by all algorithms and CPU time was observed. The results of the experiment are tabulated in Table 4.4.1. Three figures in each square of the table represent the average CPU time in seconds to solve one problem by branch and bound algorithm, heuristic-1 and heuristic-2 respectively. To conduct second type of experiment 10 problems were solved for each combination of the parameters and results are tabulated in Table 4.4.2. This table gives percentage of problems for different node-density combinations whose maximal flow value deviated from the optimal flow value by 0%, 0-5%, 5-10%, ..., 20-25% or more, when solved by heuristic-1 and heuristic-2.

Several graphs were plotted to have better under standing of the computational times. Figure 4.4.1 demonstrates the relative performance of node-disjoint algorithms on time scale and Fig. 4.4.2, 4.4.3 and 4.4.4 show the variations in computational times as nodes and density change.

4.4.1 Performance of Branch and Bound Algorithm

Total number of feasible solutions is proportional to n! and some power of d. Even though the percentage of subproblems generated decreases with increase in n or d, the number of subproblems generated increases rapidly with increase in n or d. However, small sized problems are solved very efficiently. If number of arcs in a network are less than 100 then there is no significant difference in computational time of solving the problem by branch and bound algorithm or any of the heuristics. So application of branch and bound algorithm is recommended when problem size is small.

Problems with number of arcs less than 300 are generally solved in less than 6 seconds. With a further increase in number of arcs the computational time increases rapidly. It was observed that by doubling the number of arcs of a problem with 300 arcs the computational time increased twentyfold. Still it is not the computational time which restricts the solution of large sized problems, it is the large memory requirement which put a limit on the size of the problem.

4.4.2 Performance of Heuristics

Two criteria are generally used to measure the efficiency of a heuristic method. They are computational time and accuracy of the solution.

Heuristic-1 solves a large sized problem in quite a reasonable time. Problems having 100 nodes and 2000 arcs are generally solved in less than 1 minute. It was observed that computational time increased quite linearly with increase in problem size. This method also provides quite accurate solutions. When the network is less dense ($d \leq 0.30$), then at least 80% of the problems are solved optimally. With increased values of densities more problem deviated from optimal. Even for large problem sizes at least 50% of the problems were solved optimally and about 80 of the problems were solved with at most 10% deviation of the maximal value provided by the heuristic from that of optimal value.

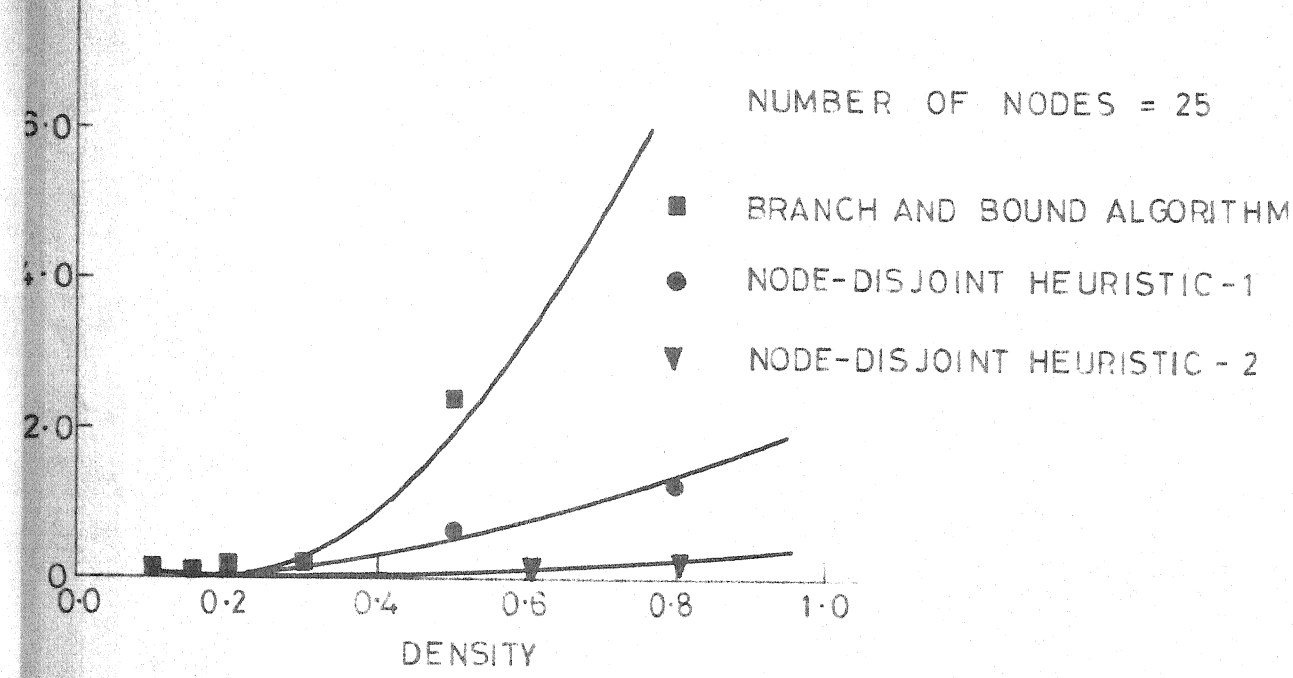
Heuristic 2 takes much less time than that of heuristic-1 but at the cost of less accuracy. Problems having 100 nodes and 2000 arcs are generally solved in less than 10 seconds, which is one sixth of the time to solve problem of same size by heuristic-1. This heuristic also exhibits linear trend in computational time with increasing density values. However, the solutions are not very accurate. Accuracy goes on diminishing with increase in density values. A trade off between computational cost and accuracy may determine the selection of appropriate heuristic.

Table 4.4.1: Computational times for node-disjoint algorithms

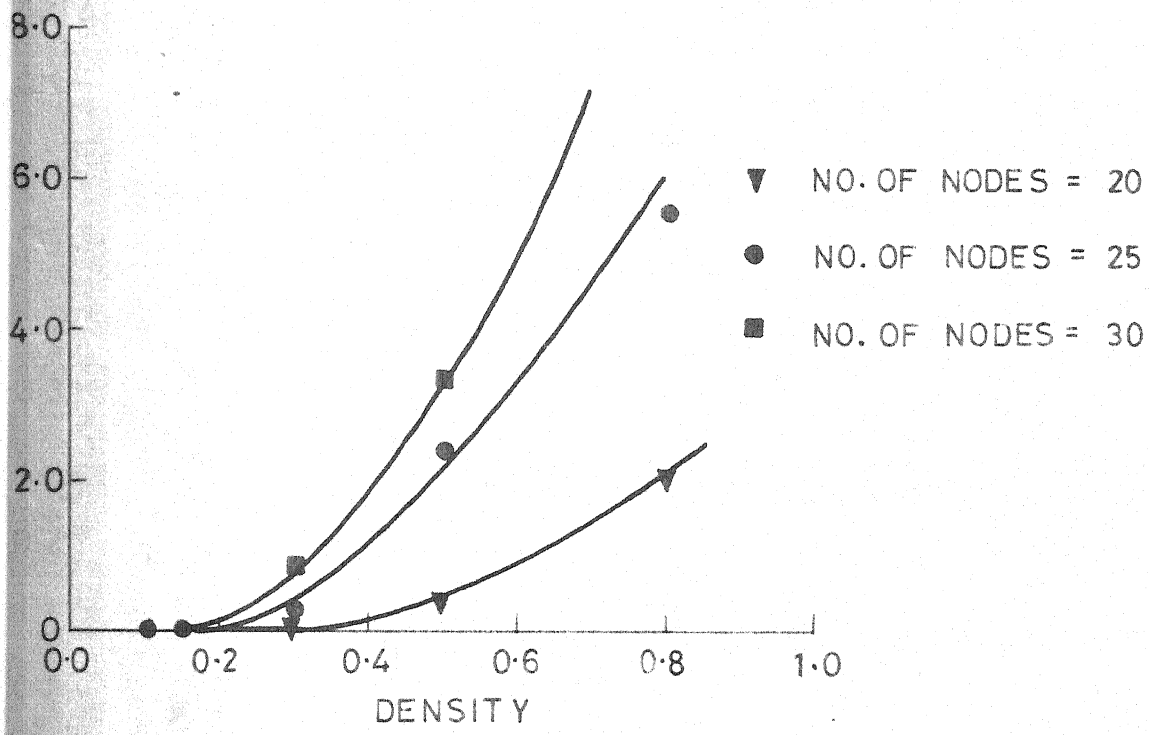
No. of nodes \ Density	0.10	0.15	0.20	0.30	0.50	0.80
10		0.01	0.01	0.02	0.02	0.06
	--	0.01	0.01	0.01	0.02	0.04
	-	0.01	0.01	0.01	0.02	0.03
15		0.02	0.02	0.03	0.13	0.51
	-	0.02	0.02	0.02	0.06	0.23
	-	0.02	0.02	0.03	0.05	0.07
20	0.03	0.03	0.05	0.09	0.41	2.12
	0.03	0.03	0.04	0.08	0.20	0.57
	0.03	0.03	0.04	0.06	0.10	0.14
25	0.04	0.05	0.10	0.23	2.40	5.50
	0.04	0.04	0.07	0.15	0.65	1.24
	0.04	0.06	0.08	0.09	0.16	0.26
30	0.07	0.10	0.18	0.84	3.20	-
	0.06	0.08	0.12	0.45	1.32	2.67
	0.06	0.09	0.11	0.17	0.26	0.46
50	0.23	-	-	-	-	-
	0.25	0.87	1.06	2.35	6.36	14.38
	0.25	0.35	0.44	0.68	1.20	1.64
80		-	-	-	-	-
	1.64	3.25	3.85	11.79	25.63	-
	0.09	1.21	1.73	4.54	4.74	-
100	-	-	-	-	-	-
	2.21	3.78	7.84	15.43	-	-
	1.83	2.61	2.67	4.65	-	-

le 4.4.2: Performance of node-disjoint heuristics.

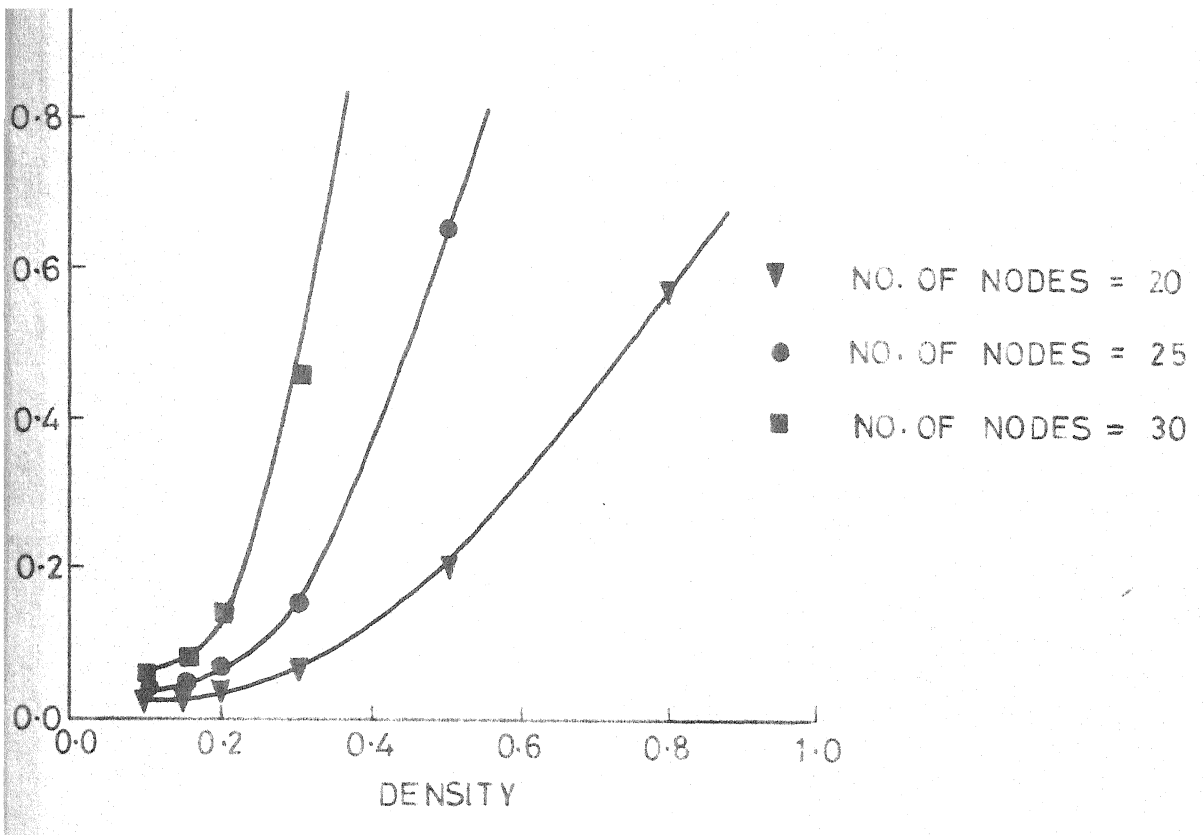
of Density	Heuristic-1 Percent Problems with deviations as: 0	5	10	15	20	25	More	Heuristic 2 Percent problems with deviation as: 0	5	10	15	20	25	More
0.2	100							100						
0.3	100							90				10		
0.4	100							60				30		10
0.5	100							70		10		10		10
0.8	80			10	10			30		10	10	20		30
0.2	100							90	10					
0.3	100							80			10			10
0.4	100							50	20	10				20
0.5	60		20	10	10			40		10		20		30
0.8	50	10	20	20					20		10	30	10	30
0.2	80	10		10				80			10			10
0.3	90		10					50		20			20	10
0.4	90			10				50			20			30
0.5	70	10		10	10			50	20			20		10
0.8	50	40			10			10		20	20	20	10	20
0.1	100							100						
0.2	90				10			90						10
0.3	90			10				30		10	20		10	30
0.4	60	40						60				10		30
0.5	40	10	20	30				10	10	10			20	50
0.1	90				10			90						10
0.2	100							60	10		30			
0.3	60	30	10					30		30	20	10		10
0.4	70	10	20					10	20	10	10	20	10	20
0.5	50	40	10						10	10	20		10	50



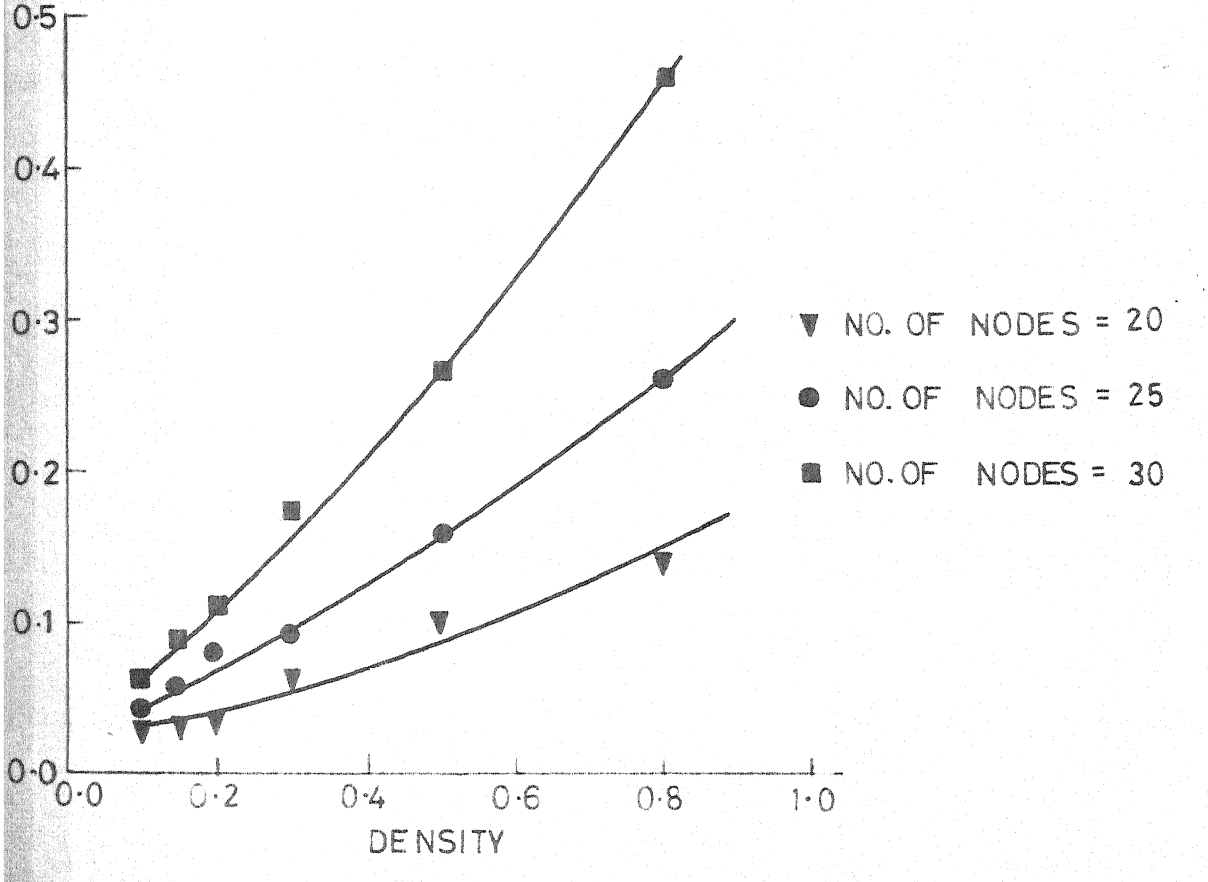
G.4.4.1 RELATIVE PERFORMANCE OF ALGORITHMS



G.4.4.2 PERFORMANCE OF BRANCH AND BOUND ALGORITHM



IG. 4·4·3 PERFORMANCE OF NODE-DISJOINT HEURISTIC-1



.4·4·4 PERFORMANCE OF NODE-DISJOINT HEURISTIC-2

CHAPTER V

MAXIMAL ARC-DISJOINT AND NODE-DISJOINT MULTICOMMODITY FLOW PROBLEMS

The problems considered in this chapter are the extensions of the problems considered in previous chapters. An attempt is made to develop methods similar to that of two-commodity flow problems for the multicommodity flow problems. Heuristics are extended for multicommodity networks, however extension of branch and bound algorithms was not attempted because branch and bound algorithms for these problems would be computationally very slow.

In this chapter two heuristics are developed for edge-disjoint multicommodity maximal flow problem and two for node-disjoint multicommodity maximal flow problem. Computational performances of the programs, built for these methods, is also presented.

5.1 MAXIMAL ARC-DISJOINT MULTICOMMODITY FLOW PROBLEM

The problem to find maximum sum of flows of commodities in a multicommodity network, such that no arc is having more than one commodity over it, is considered in this section.

The problem may be formulated along the same lines as the formulation for two-commodity arc-disjoint flow problems

modifications will be required in the constraints as there are q commodities instead of two commodities. The arc-disjoint constraints shall be the most affected one since product of arc flows is zero for each combination of the commodities. An integer programming formulation may be offered along the similar lines as that of two-commodity case.

Extension of the branch and bound algorithm for two-commodity arc-disjoint flow problem to the multicommodity arc-disjoint flow problem poses some problems. The upper bounding criteria will no more remain an efficient criteria because all unassigned arcs are assigned to each commodity and as a result the upper bound may be as bad as q times any feasible solution. Thus with increase in number of commodities, the upper bounding criteria keeps on worsening. This ultimately leads to large number of subproblem enumerated and computational time is proportionately large. Moreover, more flow values are stored at each subproblem which cuts down the size of the problem solved by this approach.

On account of these reasons no attempt was made to develop branch and bound algorithm to solve this class of problems. The success of the heuristics to solve two commodity flow problem by providing solutions very close to the optimal and in a reasonable amount of computational time tempted us to extend them to multicommodity cases. The heuristic methods,

which are described below, promise a considerable saving in computational time and provide very good solutions if not the best.

5.1.1 Heuristic-A

q single commodity maximal flow problems are solved, each from s^k to t^k, k = 1, 2, ... q. The arc having maximum number of commodities flowing over it and with maximum total flow among such arcs is chosen. Decrease in the value of the maximal flow for each commodity is calculated if the chosen arc is not included in the network. The chosen arc is finally assigned to the commodity for which decrease in the maximal flow is largest. Flow values are changed accordingly and this basic step is repeated. When no arc is left which allows more than one commodity to flow over it, the method terminates.

The step by step procedure of this heuristic follows now.

NOTATIONS

S_k : Set of arcs assigned to commodity k.

S_0 : Set of unassigned arcs.

Step 0 : Set $S_k = \phi$, k = 1, 2, ..., q,
and $S_0 = A$.
Go to step 1.

Step 1 : Solve the single commodity maximal flow problem from s^k to t^k over $G(N,A)$ for all commodities, $k = 1,2,\ldots q$. Let $X^k = \{x^k_{IJ}\}$ denote the arc flow vector and v^k the maximal flow value for commodity k. Go to step 2.

Step 2 : Using the arc flow vectors X^k, $k = 1,2,\ldots,q$, select the arc (a,b) which has maximum number of commodities flowing over it and maximum total flow among such arcs.

If no such arc is found go to step 6; otherwise go to step 3.

Step 3 : Solve the s.c. max. flow problem from s^k to t^k over $G(N,A^k)$ where $A^k = S_0 \cup S_k \quad (ab)$ for $k=1,2,\ldots,q$. Let X^k_* and v^k_* denote the respective arc flow vector and maximal flow value. Go to step 4.

Step 4 : Determine the commodity p for which decrease in maximal flow value is largest, i.e., $(v^k - v^k_*)$ is largest.

Go to step 5

Step 5 : Assign the selected arc (a,b) to commodity p.

$S_p = S_p + (ab)$

$S_0 = S_0 - (ab)$

$X^k = X^k_*$, $k = 1,2,\ldots,q$, $k \neq p$

$v^k = v^k_*$, $k = 1,2,\ldots,q$, $k \neq p$

Go to step 2.

Step 6 : The flow values X^k, v^k, $k = 1,2,\ldots,q$ are feasible. Stop.

5.1.3 Heuristic 2

The procedure starts by determining maximal augmenting paths from s^k to t^k, $k = 1,\ldots,q$. Flow is augmented along the path which has maximal augmenting capacity. All the arcs that lie on the path selected for augmentation are assigned to the commodity which flows over it. Contrary to heuristic 1 more than one arcs are assigned in one iteration. With the changed flow values this basic step is repeated again until no augmenting path exists from s^k to t^k, $k = 1,2,\ldots,q$.

Following is the detailed step by step procedure of heuristic-2.

NOTE: Sets S_k and S_o have the same meaning as in the description of heuristic-1 in 5.1.1.

Step 0 : Set $S_k = \emptyset$, $k = 1,2,\ldots,q$

$S_o = A$.

Step 1 : Determine maximal augmenting path from s^k to t^k over the network which comprises of arcs $S_k \cup S_o$ for all $k = 1,2,\ldots,q$. Let a_k denote the augmenting capacity of these paths respectively. Go to step 2.

Step 2 : If $a_k = 0$ for $k = 1,2,\ldots,q$, Go to step 5, otherwise go to step 3.

Step 3 : Determine the commodity p which has augmenting path with largest augmenting capacity, i.e., $a_p \geq a_k$, $k = 1, 2, \ldots, q$. Go to step 4.

Step 4 : Augment the flow on the maximal augmenting path of commodity p. Let S_p denote the set of arcs lying on this path. Then

$S_p = S_p \cup S_p$

$S_o = S_o \; S_p$

Go to step 1.

Step 5 : Print the flow values and stop.

5.2 MAXIMAL NODE-DISJOINT MULTICOMMODITY FLOW PROBLEM

The problem to find maximum sum of flows of commodities in a multicommodity network, such that no node is having more than one commodity flowing over it, is considered in this section.

Mathematical formulation of this problem will be along similar lines as for its special case of two-commodity networks. Along the similar lines the problem may be formulated as integer linear programming problem.

The reasons, given in Section 1, for not developing the branch and bound algorithm for arc-disjoint flow problem also hold for not developing the same for node-disjoint flow problem. Instead two heuristics are proposed. Performance of these heuristics was very encouraging for two-

commodity flow problems and it is hoped that they will also do well for multicommodity flow problems.

5.2.1 Heuristic-1

Single commodity maximal flow problems are solved from s^k to t^k for $k = 1,2,\ldots,q$. The node which allows maximum number of commodities to flow through it and has maximum total flow among such arcs is selected. Decrease in the value of the maximal flow for each commodity is calculated if the selected node and arcs incident upon it are removed from the network. The chosen node is finally assigned to the commodity for which decrease in the maximal flow value is largest. With the changed flow values this basic step is repeated. When there is no node which allows more than one commodity to flow through it, the method terminates.

The step by step procedure for this method is given below:

NOTATIONS

S_k : Set of nodes assigned to commodity k.

S_0 : Set of unassigned nodes.

Step 0 : Set $S_k = \phi$, $k = 1,2,\ldots,q$,
and $S_0 = A$.
Go to step 1.

Step 1 : Solve the s.c. max. flow problem from s^k to t^k over $G(N,A)$ for $k = 1,2,\ldots,q$. Let $X^k = x^k_{ij}$ denote the arc flow vector and v^k the maximal flow value of commodity k. Go to step 2.

Step 2 : Using the arc flow vectors X^k, $k = 1,2,\ldots,q$, select the node n_o which has maximum number of commodities flowing through it and has maximum total flow among such nodes. If no such node is found, go to step 6, otherwise go to step 3.

Step 3 : Solve the s.c. max. flow problem from s^k to t^k over the network comprising of nodes $S_k \cup S_o - n_o$ and arcs connecting this set of nodes. Do it for $k = 1,2,\ldots,q$. Let $X^k_{\#}$ and $v^k_{\#}$ denote the respective arc flow vector and maximal flow value. Go to step 4.

Step 4 : Determine the commodity p for which decrease in the maximal flow value is largest, i.e., $(v^k - v^k_{\#})$ is largest.

Go to step 5.

Step 5 : Assign the selected node n_o to commodity p.

$S_p = S_p + n_o$

$S_O = S_O - n_o$

$X^k = X^k_{\#}$, $k = 1,2,\ldots,q$, $k \neq p$

$v^k = v^k_{\#}$, $k = 1,2,\ldots,q$, $k \neq p$

Go to step 2.

Step 6 : Print the flow values X^k, v^k, $k = 1,2,\ldots,q$ and stop.

5.2.2 Heuristic 2

The procedure starts by determining maximal augmenting paths from s^k to t^k, $k = 1,2,\ldots,q$. Flow is augmented along the path which has maximum augmenting capacity. All the nodes that lie on the path selected for augmentation are assigned to the commodity which flows over it. Hence many nodes are assigned in one iteration. With the changed flow values, this step is repeated. When no augmenting path exists from s^k to t^k, $k = 1,2,\ldots,q$, the procedure terminates.

The stepwise procedure now follows.

NOTATION

S_k : Set of nodes assigned to commodity k.

S_0 : Set of unassigned nodes.

Step 0 : Set $S_k = \emptyset$, $k = 1,2,\ldots,q$,
and $S_0 = N$. Go to step 1.

Step 1 : Determine maximal augmenting path from s^k to t^k over the network which comprises of nodes $S_k \cup S_0$ and arcs connecting this set of nodes. Do it for $k=1,2,\ldots,q$. Let a_k denote the augmenting capacity of maximal augmenting path of commodity k.
Go to step 2.

Step 2 : If $a_k = 0$ for all $k = 1,2,\ldots,q$, Go to step 5; otherwise go to step 3.

computational time, which was in terms of CPU time, was observed. As a result tables were generated which provide average CPU time to solve one problem for different node-density combinations.

Each node-density combination corresponds to one square in the table. Each square contains three figures which are the computational times to solve problems having 2, 5 and 8 commodities respectively. Tables 5.3.1 and 5.3.2 present the computational times in seconds for edge-disjoint heuristic 1 and heuristic-2 respectively. Computational times of node-disjoint heuristic-1 and heuristic-2 are tabulated in Tables 5.3.3 and 5.3.4.

Several graphs were plotted to facilitate better understanding of the computational times. In Figure 5.3.1 computational times of all the heuristics are plotted against density. Figure 5.3.2 depicts the relationship between computational time and density for various number of commodities, whereas Figure 5.3.3 shows the variation in computational time as density and number of nodes are changed. These two graphs are plotted from the table of edge-disjoint heuristic-2; however, same pattern is observed for other heuristics also.

Due to unavailability of exact algorithms for multi-commodity disjoint flows, it was not possible to measure the

deviations of the solutions provided by the heuristics from that of optimal ones. However, relative performances of the heuristics was judged.

5.3.1 Results and Discussions

One observation which is evident from Fig. 5.3.1 is that heuristic 1 takes considerably more computational time than heuristic-2, be it a arc-disjoint flow or node-disjoint flow. Moreover this difference in computational times keeps on increasing as problem size increases. This drawback of heuristic-1 is counterbalanced by the fact that generally it provides better solutions. The question, that which of the two is better, does not have a straightforward answer. A tradeoff between accuracy and cost might determine the selection of appropriate heuristic.

It was mentioned in chapter 4 that transformation of node-disjoint flow problems to arc-disjoint flow problem may not be economical. This is evident from Fig. 5.3.1 that node-disjoint heuristics takes considerably less time than their arc-disjoint counterparts. Hence, we have been right in our decision to develop separate algorithms for them.

Table 5.3.1: Computational performance of edge-disjoint heuristic-1.

No. of nodes \ Density	0.10	0.15	0.20	0.30	0.50	0.80
10		-	0.02	0.02	0.02	0.10
	-	-	0.02	0.03	0.07	0.19
	-	-	0.03	0.03	0.09	0.35
20	0.03	0.03	0.04	0.06	0.14	0.95
	0.05	0.04	0.06	0.26	1.64	9.44
	0.04	0.05	0.12	0.53	3.72	11.72
40	0.12	0.15	0.46	0.75	1.75	7.01
	0.23	1.38	4.46	24.20	76.00	226.83
	0.24	1.38	5.44	37.48	131.77	307.62
60	0.35	1.37	1.48	2.49	5.35	-
	2.43	14.93	29.45	197.35	498.17	
	7.15	22.96	56.10	314.26	-	-
80	0.90	2.02	1.63	-	-	-
	30.39	90.99	257.11	-	-	-
	55.79	105.25	353.32	-	-	-
100	1.94	2.04		-	-	-
	54.89	94.85	-	-	-	-
	95.12	213.55	-	-	-	-

Table 5.3.2: Computational performance of edge-disjoint heuristic-2

No. of Nodes \ Density	0.10	0.15	0.20	0.30	0.50	0.80
10	-	-	0.01	0.02	0.03	0.05
	-	-	0.02	0.03	0.06	0.12
	-	-	0.03	0.04	0.09	0.22
20	0.05	0.04	0.04	0.10	0.18	0.35
	0.04	0.06	0.13	0.21	0.57	1.29
	0.07	0.09	0.13	0.40	1.08	1.93
40	0.17	0.23	0.44	0.60	1.41	2.55
	0.52	0.81	1.40	2.68	5.06	12.54
	0.44	1.11	2.41	5.64	11.97	27.44
60	0.52	1.18	1.24	2.55	4.64	-
	1.86	3.58	3.35	10.96	20.03	-
	3.82	6.50	12.32	20.45	44.14	-
80	1.20	2.67	3.07	4.44	-	-
	6.39	9.83	12.65	28.40	-	-
	11.14	19.76	34.02	68.51	-	-
100	2.99	3.71	4.91	-	-	-
	10.70	17.95	23.45	-	-	-
	20.22	28.55	62.38		-	-

Table 5.3.3: Computational performance of node-disjoint heuristic-1.

No. of Nodes \ Density	0.10	0.15	0.20	0.30	0.50	0.80
10			0.01	0.01	0.02	0.06
		-	0.02	0.02	0.05	0.08
		-	0.03	0.03	0.06	0.12
20	0.03	0.03	0.05	0.08	0.15	0.67
	0.03	0.04	0.06	0.22	0.58	1.54
	0.04	0.07	0.12	0.29	0.85	1.94
40	0.12	0.14	0.55	1.25	2.95	7.33
	0.19	0.71	1.51	4.07	9.46	21.80
	0.20	0.72	1.63	7.01	13.09	28.05
60	0.38	2.12	1.48	3.60	10.48	-
	1.52	4.23	9.28	23.80	43.83	-
	2.78	7.71	14.09	31.68	53.31	-
80	1.37	4.82	4.07	14.87	-	-
	9.68	18.70	36.15	65.14	-	
	11.62	25.61	49.11	94.47	-	-
100	5.10	5.38	4.62	-	-	-
	24.98	54.61	93.23		-	-
	32.22	56.64	94.31		-	-

Table 5.3.4: Computational performance of node-disjoint heuristic-2.

Density / No. of Nodes	0.10	0.15	0.20	0.30	0.50	0.80
10		-	0.01	0.01	0.02	0.03
		-	0.02	0.02	0.04	0.06
		-	0.03	0.03	0.05	0.08
20	0.05	0.04	0.04	0.06	0.13	0.16
	0.05	0.05	0.09	0.11	0.20	0.31
	0.07	0.07	0.10	0.15	0.26	0.36
40	0.16	0.20	0.25	0.40	0.65	1.07
	0.24	0.41	0.55	0.71	1.03	1.66
	0.27	0.49	0.82	0.96	1.30	2.14
60	0.47	0.85	0.89	1.58	2.09	-
	1.05	1.40	1.56	2.42	3.19	-
	1.39	1.64	2.47	3.09	4.25	-
80	0.98	1.37	2.04	3.07	-	-
	2.18	3.13	3.36	5.08	-	-
	3.07	3.86	5.03	11.18	-	-
100	1.33	2.80	3.94	-	-	-
	1.99	4.43	8.78	-	-	-
	5.36	8.00	7.99	-	-	-

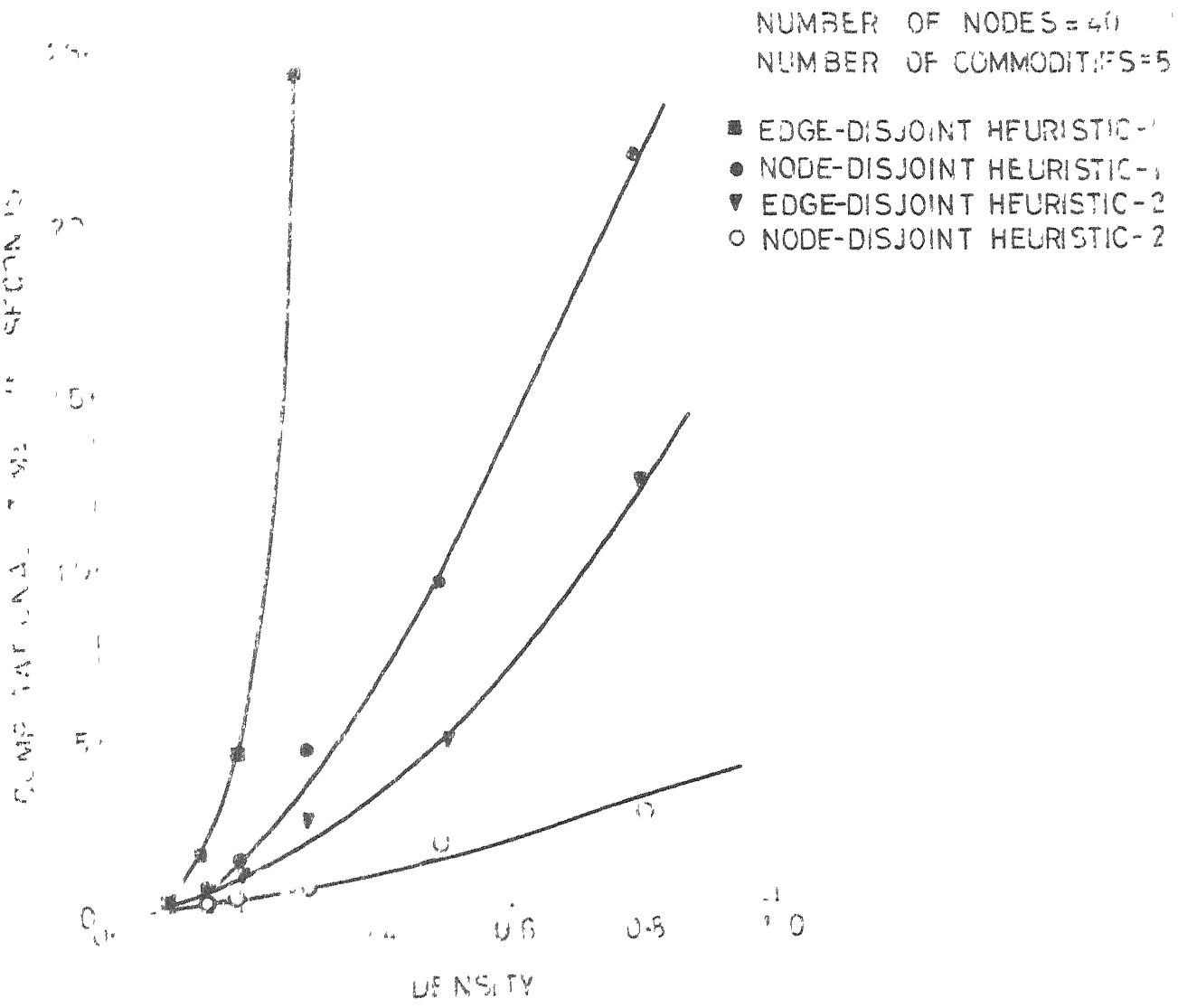


FIG. 5.3.1 COMPARISON OF HEURISTICS

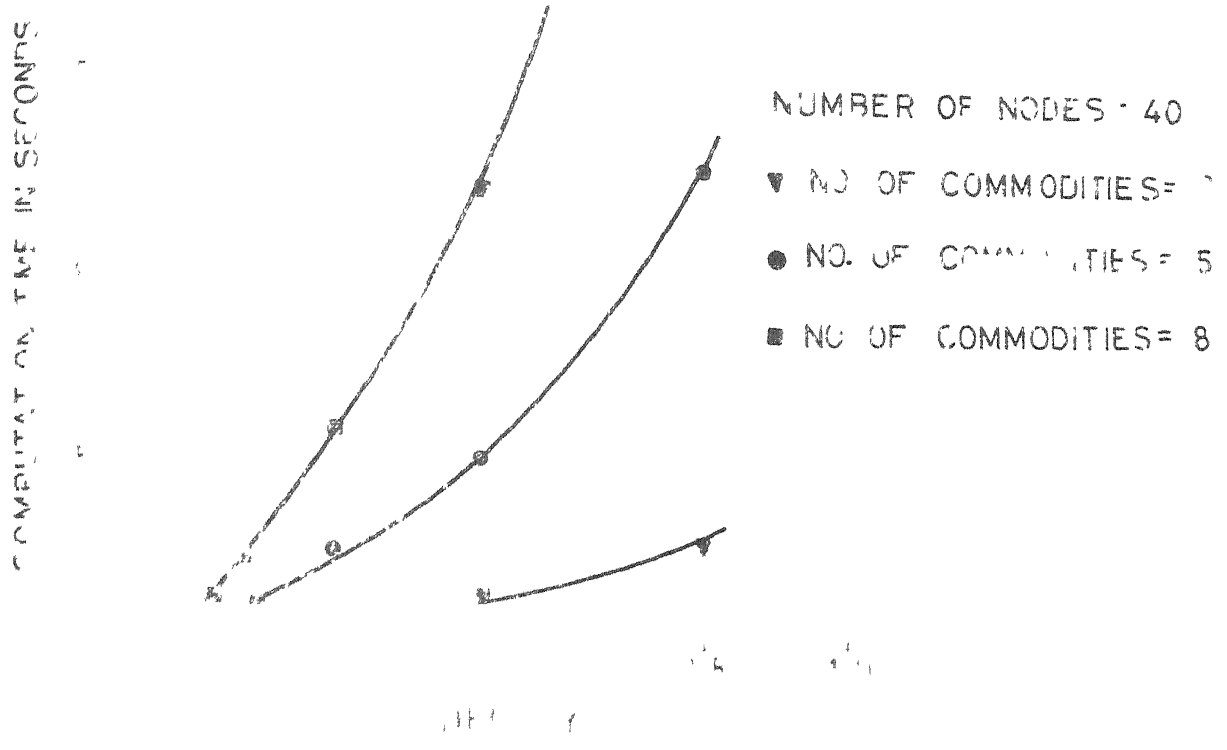


FIG. 5 VARIATION OF COMP. TIME WITH NO. OF COMMODITIES

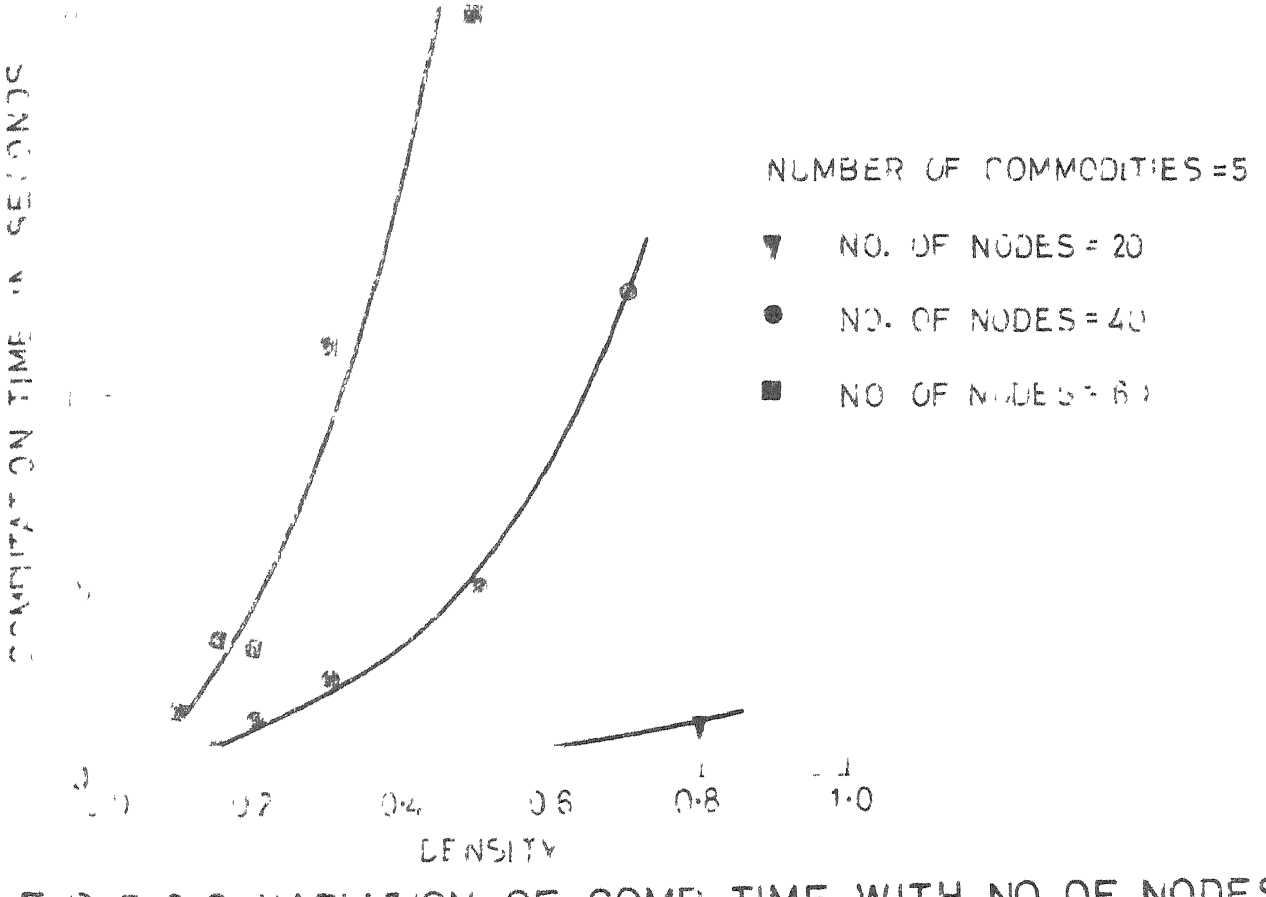


FIG 5·3·3 VARIATION OF COMP. TIME WITH NO. OF NODES

CHAPTER VI

AVENUES FOR FURTHER RESEARCH

In this work an attempt has been made to propose solution techniques for a class of disjoint flow problems. The proposed methods are not perfect, however they can be credited for initiating research in this field. Methods may yield better results by introducing some intelligent modifications. For example, a different upper bounding criteria may be suggested for branch and bound algorithms. Similarly much better performance can be expected from the computer programs if they are suitably modified.

Development of exact methods for multicommodity disjoint flows should be the concern of the persons interested in this field.

The present work was concerned with pure disjoint flows, i.e., where all arcs or nodes allow only one commodity to flow over it. It may be worthwhile to consider the mixed disjoint flows, i.e., where some arcs or nodes allow simultaneous flow of commodities.

REFERENCES

1. Assad, A. ..., 'Multicommodity Network Flows - A Survey', Networks, Vol. 8, No. 1, pp 37-91, 1978.

2. Chein, L. a... C. Dewald, 'A Generalized Chain Labelling Algorithm for Solving Multicommodity Flow Problems', Comp. and Opns. Res., Vol. 1, pp 437-465, 1974.

3. Cremeans, J.E., R. Smith and G. Tyandall, 'Optimal Multicommodity Flows with Resource Allocation' NRLQ, Vol. 17, pp 269-280, 1970.

4. Dinic, E.A., 'Algorithm for Solution of a Problem of Maximal Flow in a Network with Power Estimation', Sovt. Math. Dokl., Vol. 11, No. 5, pp 1277-1280, 1970.

5. Dreyfus, S. 'An Appraisal of Some Shortest Path Algorithms' Opns. Res., Vol. 17, No. 3, pp 395-412, 1969.

6. Evans, J.R., 'The Simplex Method for Integral Multicommodity Networks', NRLQ, Vol. 25, No. 1, pp 31-38, 1978.

7. Evans, J.R., 'A Single Commodity Transformation for Multicommodity Networks', Opns. Res., Vol. 26, No. 4, pp 673-680, 1978

8. Ford, L. R. and D. R. Fulkerson, 'Flows in Networks', Princeton University Press, Princeton, N.J., 1962.

9. Ford, L.R. and D.R. Fulkerson, 'A suggested Computation for Maximal Multicommodity Network Flows', Man. Sc., Vol. 5, No. 1, pp 97-101, 1958.

10. Geoffrion, A.M., 'Primal Resource-Directive Approaches for Optimizing Nonlinear Decomposable Systems', Opns. Res., Vol. 18, pp 373-403, 1970.

11. Golden, B.L. and T. L. Magnanti, 'Deterministic Network Optimization', Networks, Vol. 7, No. 2, pp 152-179, 1977.

12. Graves, G.W. and R. D. Bridge, 'The Factorization Approach to Large Scale Linear Programming', Math. Progr., Vol. 10, pp. 91-110, 1976.

13. Grigoriadis, M.D. and W. W. White, 'A Partitioning Algorithm for the Multicommodity Network Flow Problem', Math. Progr., Vol. 3, No. 2, pp. 157-177, 1972.

14. Grinold, R.C., 'A Multicommodity Max-Flow Algorithm', Opns. Res., Vol. 26, pp 1234-1238, 1968.

15. Held, M., P. Wolf and H. Crowder, 'Validation of Subgradient Optimization', Math. Progr., Vol. 6, pp 62-88, 1974.

16. Hu, T.C., 'Integer Programming and Network Flows', Addison Wesley, Reading, Mass., 1969.

17. Iri, M., 'On an Extension of the Maximum-Flow Minimum-Cut Theorem to Multicommodity Flows', J. Opns. Res. Soc. of Japan, Vol. 13, 129-135, 1971.

18. Itai, Alon, 'Two Commodity Flow', J. of ACM, Vol. 25, No. 4, 596-611, 1978.

19. Jarvis, J.J., 'On the Equivalence Between Node-arc and Arc-chain Formulation for the Multicommodity Flow Problem', NRLQ, Vol. 16, 525-529, 1969.

20. Jarvis, J. J. and J. B. Tindall, 'Minimal Disconnecting Sets in Directed Multicommodity Networks', NRLQ, Vol. 19, 681-690, 1972.

21. Karzanov, A., 'Determining the Max Flow in a Network by the Method of Preflows', Sovt. Math. Dokl., Vol. 15, pp 434-437, 1974.

22. Kennington, J.L., 'A Survey of Linear Cost Multicommodity Network Flows', Opns. Res., Vol. 26, No. 2, pp 209-236, 1978.

23. Kennington, J.L., 'Solving Multicommodity Transportation Problem Using a Primal Partitioning Simplex Technique', NRLQ, Vol. 24, No. 2, pp 309-326, 1977.

24. Kennington, J.L. and M. Shalaby, 'An Effective Subgradient Procedure for Minimal Cost Multicommodity Flow Problems', Man. Sc., Vol. 23, No. 9, pp 994-1004, 1977.

25. Kleitman, D.J., 'An Algorithm for Certain Multicommodity Flow Problems', Networks, Vol. 1, pp 75-90, 1971.

26. Lasdon, L.S., 'Optimization Theory for Large Scale Systems', MacMillan Co., New York, 1970.

27. Robacker, J.T., 'Notes on Linear Programming: Part XXXVII Concerning Multicommodity Networks', RM 17-99. The Rand Corporation. Santa Monica. Calif. 1956.

28. Rothfarb R. and I.T. Frisch, 'On the 3-Commodity Flow Problem', SIAM J. of Appl. Math., Vol. 17, pp 46-48, 1968.

29. Rothfarb R., H.P. Shein and I.T. Frisch, 'Common Terminal Multicommodity Flow', Opns. Res., Vol. 16, pp 202-205, 1968.

30. Rothschild, R., A. Whinston and J. Kent, 'Computing Two Commodity Flows', Opns. Res., Vol. 16, pp 446-450, 1968.

31. Swoveland, C., 'Decomposition Algorithm for the Multi-commodity Distribution Problem', Working Paper No. 184, West. Manag. Sc. Inst., Univ. of Calif., Los Angeles, 1971.

32. Tang, D.T., 'Comments on Feasibility Conditions of Simultaneous Flows in Networks', Opns. Res., Vol. 13, pp 143-146, 1965.

33. Tomlin, J.A., 'Minimum Cost Multicommodity Network Flow', Opns. Res., Vol. 14, pp 45-51, 1966.

34. Weigel, H.S. and J.E. Cremeans, 'The Multicommodity Network Flow Model Revised to Include Vehicle per Time Period and Node Constraints', NRLQ, Vol. 19, pp 77-89, 1972.

35. Wollmer, R.D., 'Multicommodity Networks with Resource Constraints - The Generalized Multicommodity Flow Problem', Networks, Vol. 1, pp 245-263, 1972.

36. Wong, R.T., 'A Survey of Network Design Problems', Working Paper OR 053-76, Opns. Res. Center, M.I.T., May 1976.

APPENDIX A

A few definitions and methods, which are used in the algorithms of the previous chapters, are described here.

A.1 SINGLE COMMODITY MAXIMAL FLOW ALGORITHM

This algorithm finds maximal flow from a source node to a sink node in a capacitated network. Algorithm proposed by Ford and Fulkerson [8] is used in this work and described here briefly. For other algorithms for the same problem see [4,21].

The algorithm begins with an initial feasible flow vector X. Some known feasible flow vector or X=0 can be used as the initial flow vector. Let $\bar{X}$ be the present feasible flow vector. The steps to follow may logically be divided into two routines: labelling routine and flow improvement routine. Labelling routine is a systematic method for checking whether a flow augmenting path with respect to $\bar{X}$ exists or not. Flow improvement routine is the routine to improve the flow on the augmenting path discovered by the labelling routine.

Labelling Routine : Label the source as (s^+,∞) and continue labelling by the following labelling rules:

(1) If i is a labelled node and j is an unlabelled node at this stage and $(i,j) \in A$ and $x_{ij} < c_{ij}$, label j with (i^+, e_j) where $e_j = \min [e_i, c_{ij} - x_{ij}]$.

(2) If i is a labelled node and j is an unlabelled node at this stage and $(j,i) \in A$ and $x_{ji} > 0$, label j with (i^-, e_j) where $e_j = \min [e_i, x_{ji}]$.

Flow Improvement Routine : Assume that sink is labelled as (k^+, e_t). Replace x_{kt} by $x_{kt} + \varepsilon_t$ and turn to N_k. If N_k is labelled as (j^-, e_k), replace x_{kj} by $x_{kj} - e_t$ and turn to N_j. Continue until source is reached. Erase all the labels and start labelling routine.

Labelled nodes are scanned to label the unlabelled nodes in the order they are labelled. When sink node is labelled, the labelling routine terminates and flow improvement routine begins. If all the labelled nodes are scanned and sink is still unlabelled, the algorithm stops. The present flow is the optimal flow.

A.2 MAXIMAL AUGMENTING PATH ALGORITHM

Maximal augmenting path in a network is the augmenting path on which maximum amount of flow can be augmented. The flow value that can be augmented on a path is known as the augmenting capacity of that path. The method to find such a path is similar to Dijkstra's method [5] to solve shortest path problems.

Classify the source node as node with permanent label and assign it a label $+\infty$. All other nodes have temporary label $-\infty$. Let source node be the permanently labelled node under consideration.

Step 1 : Change the tentative labels μ_j's of the temporarily labelled nodes j employing the following scheme, where i is the permanently labelled node under consideration with label μ_i:

$$\mu_j = \begin{cases} \max [\mu_j, \min (\mu_i, c_{ij}-x_{ij})], & \text{if } (i,j) \in A \\ \max [\mu_j, \min (\mu_i, x_{ji})], & \text{if } (j,i) \in A \\ \mu_j \text{ otherwise} \end{cases}$$

Go to step 2.

Step 2 : Determine the largest of all tentative labels and classify that node as permanently labelled node. Let this node now be the node under consideration. Go to step 1.

When, after at most (n-1) executions of these fundamental iterative steps, sink is permanently labelled the procedure terminates. Backtracking provide the maximal augmenting path and label of the sink node is the augmenting capacity of this path.

A.3 BRANCH AND BOUND APPROACH

Branch and bound approach is a very useful tool for solving combinatorial optimization problems. It provides a systematic method to search for the optimal feasible solution by doing only partial enumeration. In the course of applying the branch and bound approach the overall set of feasible solutions is partitioned into many simpler subsets. At each stage one promising subset is chosen and an effort is made to find the best feasible solution from it. If it is found then that subset is said to be fathomed. If the best feasible solution is not found then that subset is again partitioned into two or more simpler subsets (this operation known as branching) and the same process is repeated again. At each partitioning of a set into its subsets a lower bound (or upper bound) which gives the minimum value (or maximum value) of the objective function which any feasible solution in that subset may yield. If this lower bound is greater than (or lower than) some best known feasible solution (known as incumbent) then that subset is not further branched (this operation known as pruning). When all subsets are either fathomed or pruned, incumbent provides the value of an optimal solution.

```
C     ***************************************************************
C
C     THIS PROGRAM GENERATES A CONNECTED NETWORK
C
C     ***************************************************************
      SUBROUTINE RANDOM(NUMBER,NODENO,DENSTY, ARCNO,SOURSE,SINK,START,
     1 END,CAP)
C
C     NODENO : NUMBER OF NODES IN THE NETWORK
C     NUMBER : NUMBER OF COMMODITIES FLOWING
C     ARCNO : NUMBER OF ARCS IN THE NETWORK
C     DENSTY : A NUMBER WHICH CONTROLS THE DENSITY OF THE NETWORK
C     SOURSE(I) : SOURSE NODE OF COMMODITY I
C     SINK(I): SINK NODE OF COMMODITY I
C     START(J) : STARTING NODE OF ARC J
C     END(J) : ENDING NODE OF ARC J
C     CAP(J) : CAPACITY OF ARC J
C     CONECT : THE NODE-NODE INCIDENCE MATRIX GENERATED BY THE PROGRAM
C
      INTEGER CONECT(100,100),LIST(100),START(2000),END(2000),
     1 CAP(2000),SOURSE(10),SINK(10),RAND1,RAND2,TEMP1,ARCNO
      K=0
      L=0
      LISTNO=0
C***** THIS PART GENERATES THE NODE-NODE INCIDENCE MATRIX
      DO 10 I=1,NODENO
      DO 5 J=1,NODENO
5     CONECT(I,J)=0
10    LIST(I)=0
      DO 15 I=3,NODENO
      IF(RAN(X).GT.DENSTY)GO TO 15
      L=L+1
      CONTINUE
      DO 20 J=2,L
      CONECT(1,J)=1
      CONECT(J,1)=-1
      LISTNO=LISTNO+1
      LIST(J)=1
      CONTINUE
      DO 25 I=2,NODENO
      KODE=0
      DO 25 J=1,NODENO
      IF(J.EQ.I.OR.CONECT(I,J).NE.0)GO TO 25
      IF(KODE.EQ.1)GO TO 30
      IF(LISTNO.EQ.NODENO.OR.LIST(J).EQ.1)GO TO 30
      KODE=1
      LISTNO=LISTNO+1
      IF(RAN(X).GT.DENSTY)GO TO 25
```

```
35      CONECT(I,J)=1
        CONECT(J,I)=-1
25      CONTINUE
        DO 114 J=1,NODENO
11      LIST(J)=J
        DO 40 J=1,NODENO
        RAND1=RAN(X)*NODENO
        RAND2=RAN(X)*NODENO
        IF(RAND1.EQ.0.OR.RAND2.EQ.0)GO TO 40
        TEMP1=LIST(RAND1)
        LIST(RAND1)=LIST(RAND2)
        LIST(RAND2)=TEMP1
40      CONTINUE
C***** THIS PART FINDS POSITIONS OF SOURSES AND SINKS
50      DO 55 I=1,NUMBER
52      SOURSE(I)=RAN(X)*NODENO
        SINK(I)=RAN(X)*NODENO
        IF(SOURSE(I).EQ.0.OR.SINK(I).EQ.0.OR.SOURSE(I).EQ.SINK(I))GO TO 52
55      CONTINUE
C***** ARCS ARE IDENTIFIED FROM THE INCIDENCE MATRIX
        DO 60 I=1,NODENO
        DO 60 J=1,NODENO
        IF(CONECT(I,J).LE.0)GO TO 60
        K=K+1
        START(K)=LIST(I)
        END(K)=LIST(J)
60      CONTINUE
        ARCNO=K
C***** GENERATE CAPACITIES FOR ARCS
        DO 70 J=1,ARCNO
        CAP(J)=(RAN(X)*100)
70      CONTINUE
        RETURN
        END
```